21世纪高等学校计算机基础实用规划教材

ASP.NET Web 应用开发技术



喻钧 白小军 主编





清华大学出版社

21世纪高等学校计算机基础实用规划教材
普通高等教育“十二五”规划教材

ASP. NET Web应用开发技术

喻钧　白小军　主编

清华大学出版社
北京

内 容 简 介

本书从 ASP.NET 初学者的角度出发，对 ASP.NET Web 开发技术进行了由浅入深的详细介绍。全书共 12 章，分别介绍了 Web 程序设计基础、HTML 和 XML、CSS、JavaScript 脚本语言、C# 语法、服务器端控件、Web 数据库、数据绑定、Web Service、AJAX 技术等内容。

全书内容翔实，通俗易懂，适合自学。书中给出了丰富的实例以帮助读者深入理解和学习，在每章的后面还配有习题和上机练习。本书可作为高等院校学生的教材，也可作为 ASP.NET 动态网页设计人员的参考书。

图书在版编目(CIP)数据

ASP. NET Web 应用开发技术/喻钧，白小军主编. —北京：清华大学出版社，2012. 11
ISBN 978-7-302-29830-4

Ⅰ. ①A…　Ⅱ. ①喻…　②白…　Ⅲ. ①网页制作工具—程序设计　Ⅳ. ①TP393. 092

中国版本图书馆 CIP 数据核字(2012)第 197187 号

责任编辑： 魏江江　赵晓宁
封面设计： 傅瑞学
责任校对： 白　蕾
责任印制：

出版发行： 清华大学出版社
　　网　　址： http://www. tup. com. cn，http://www. wqbook. com
　　地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
　　社 总 机： 010-62770175　　**邮　　购：** 010-62786544
　　投稿与读者服务： 010-62776969，c-service@tup. tsinghua. edu. cn
　　质量反馈： 010-62772015，zhiliang@tup. tsinghua. edu. cn
　　课件下载： http://www. tup. com. cn，010-62795954
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185mm×260mm　**印　张：** 20　　**字　　数：** 489 千字
版　　次： 2012 年 11 月第 1 版　　**印　　次：** 2012 年 11 月第 1 次印刷
印　　数： 1～　000
定　　价： 　.00 元

产品编号：041284-01

编审委员会成员

（按地区排序）

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施"高等学校本科教学质量与教学改革工程(简称'质量工程')"，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部"质量工程"的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程(群)加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合"质量工程"的实施，满足各高校教学质量和教学改革的需要。

本系列教材立足于计算机公共课程领域，以公共基础课为主、专业基础课为辅，横向满足高校多层次教学的需要。在规划过程中体现了如下一些基本原则和特点。

(1) 面向多层次、多学科专业，强调计算机在各专业中的应用。教材内容坚持基本理论适度，反映各层次对基本理论和原理的需求，同时加强实践和应用环节。

(2) 反映教学需要，促进教学发展。教材要适应多样化的教学需要，正确把握教学内容和课程体系的改革方向，在选择教材内容和编写体系时注意体现素质教育、创新能力与实践能力的培养，为学生的知识、能力、素质协调发展创造条件。

(3) 实施精品战略，突出重点，保证质量。规划教材把重点放在公共基础课和专业基础课的教材建设上；特别注意选择并安排一部分原来基础比较好的优秀教材或讲义修订再版，逐步形成精品教材；提倡并鼓励编写体现教学质量和教学改革成果的教材。

(4) 主张一纲多本，合理配套。基础课和专业基础课教材配套，同一门课程可以有针对不同层次、面向不同专业的多本具有各自内容特点的教材。处理好教材统一性与多样化，基本教材与辅助教材、教学参考书，文字教材与软件教材的关系，实现教材系列资源配套。

(5) 依靠专家,择优选用。在制定教材规划时依靠各课程专家在调查研究本课程教材建设现状的基础上提出规划选题。在落实主编人选时,要引入竞争机制,通过申报、评审确定主题。书稿完成后要认真实行审稿程序,确保出书质量。

繁荣教材出版事业,提高教材质量的关键是教师。建立一支高水平教材编写梯队才能保证教材的编写质量和建设力度,希望有志于教材建设的教师能够加入到我们的编写队伍中来。

21世纪高等学校计算机基础实用规划教材

联系人:魏江江 weijj@tup.tsinghua.edu.cn

前　言

ASP.NET 是在 Microsoft .NET Framework 的基础上构建的、可提供构建企业级 Web 应用程序所需的 Web 平台，是创建动态交互网页的强有力的工具。.NET Framework 是用于构建、开发以及运行 Web 应用程序和 Web Service 的公共环境，主要由编程语言、服务器端和客户端技术、开发环境三部分组成。ASP.NET 是.NET Framework 的重要组成部分，它建立在公共语言运行库上，可用于在 Web 服务器上生成功能强大的 Web 应用程序，为 Web 站点创建动态的、交互的 HTML 页面。

作为企业级应用开发的两大主流技术体系之一，.NET 技术近年来发展异常迅速，越来越受到国内外 IT 企业的认可，在各行各业都得到了广泛的应用。因此，对.NET 研发人员的需求量也在不断上升，熟悉.NET 技术体系的学生就业前景很好。

笔者长期从事 Web 应用开发和.NET 技术课程的一线教学工作，有着深厚的实践开发经验和丰富的教学经验，熟悉 ASP.NET 和 JAVA EE 两大主流技术体系，对面向对象技术、设计模式、软件架构等知识理解较为深刻，能够站在理论的高度来指导实践。同时，笔者非常了解学生的认知规律，并以此指导教材的编写。

Web 应用开发有着很强的技巧性，要求学生从整体上把握软件的架构、框架，合理地使用设计模式，这样才能设计稳定性好、扩展性强的软件产品。本教材更注重思想方法的培养，将面向对象思想、设计模式和软件架构的知识融入各章节教学中，尽量使学生知其然并知其所以然，以思想方法指导设计实践。本书紧跟技术潮流，反映了 Web 2.0 时代的技术特征；同时没有完全跟风，而是选择主流、稳定的技术进行讲解，保证了内容体系的稳定性。

全书所有程序在 Windows XP SP2/Windows 7、IIS 6.0/7.0、.NET Framework 4 下测试通过，数据库使用 SQL Server 2008 R2 版，开发工具采用 Microsoft Visual Studio 2010 旗舰版。

本书第 1～第 3、第 5～第 8 和第 11 章由喻钧编写，第 4、第 9、第 10 和第 12 章由白小军编写，全书由喻钧统稿。本书电子课件和源程序可以在清华大学出版社网站 http://www.tupwk.com.cn/downpage 免费下载。

在编写本书的过程中，得到了西安凌安电脑有限公司张西京经理及其同仁的大力支持，在此一并表示感谢。尽管在编写本书的过程中尽了最大努力，但由于笔者水平有限，疏漏及不妥之处在所难免，恳请读者批评指正。作者联系邮箱：jyu0117@163.com 。

编　者

2012 年 8 月

目　录

第 1 章　Web 程序设计基础

Web 技术的迅速发展极大地改变了整个世界，影响着人们生活、工作、学习、交流的方方面面。Web 已经构成了 Internet 上最强有力的信息检索和查询工具。Web 应用结构包括装有浏览器软件的客户端、Web 服务器，以及连接它们的通信网络。客户端与 Web 服务器之间遵循 HTTP 协议进行请求和响应。当今多数 Web 站点都是具有复杂交互能力的 Web 页面，这些动态 Web 页面(也称为 Web 应用程序)集成了网页信息和处理程序，位于 Web 服务器上，可以根据用户的不同请求进行处理，最后将结果返回给客户端。

本章向读者介绍 Web 程序设计的最基础知识，包括软件编程体系、Web 的工作原理以及 Web 程序设计技术等。

1.1　软件编程体系

在目前的应用软件开发领域中，主要分成两大软件编程体系结构：一种是客户机/服务器(Client/Server，C/S)结构；另一种是浏览器/服务器(Browser/Server，B/S)结构。这两种结构所支持的语言开发工具是不同的，如图 1-1 所示。

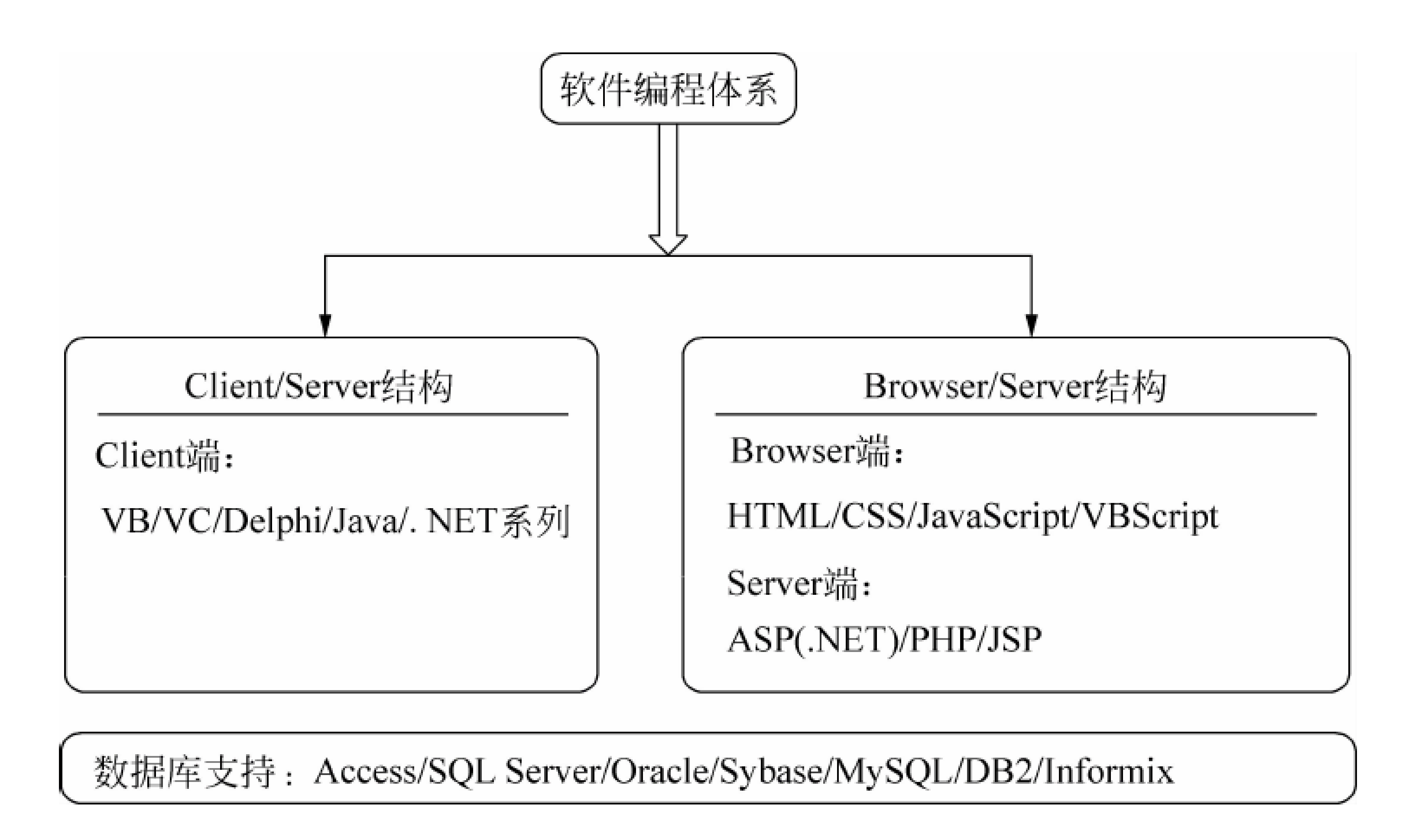


图 1-1　软件编程体系结构

1.1.1 C/S 软件体系结构

C/S 是较早的软件系统体系结构，主要用于局域网环境，如图 1-2 所示。服务器通常采用高性能的 PC 或工作站，并安装大型的数据库系统，如 Oracle、Sybase、Informix 或 SQL Server，也称为数据库服务器。客户机需要安装专门的客户端软件，即应用程序。这种结构能够充分发挥客户端 PC 的处理能力，客户端响应速度快。C/S 系统的可扩展性和可维护性差，如软件升级时，每台客户机都需要重新安装，系统维护和升级的成本高；同时，任一台客户端的应用程序出问题，都不能正常工作，需要重新安装和维护。另外，由于采用 Intranet 技术，适用于局域网环境的可连接用户数有限，当用户数量增多时，系统性能会明显下降，代码的可重用性差。

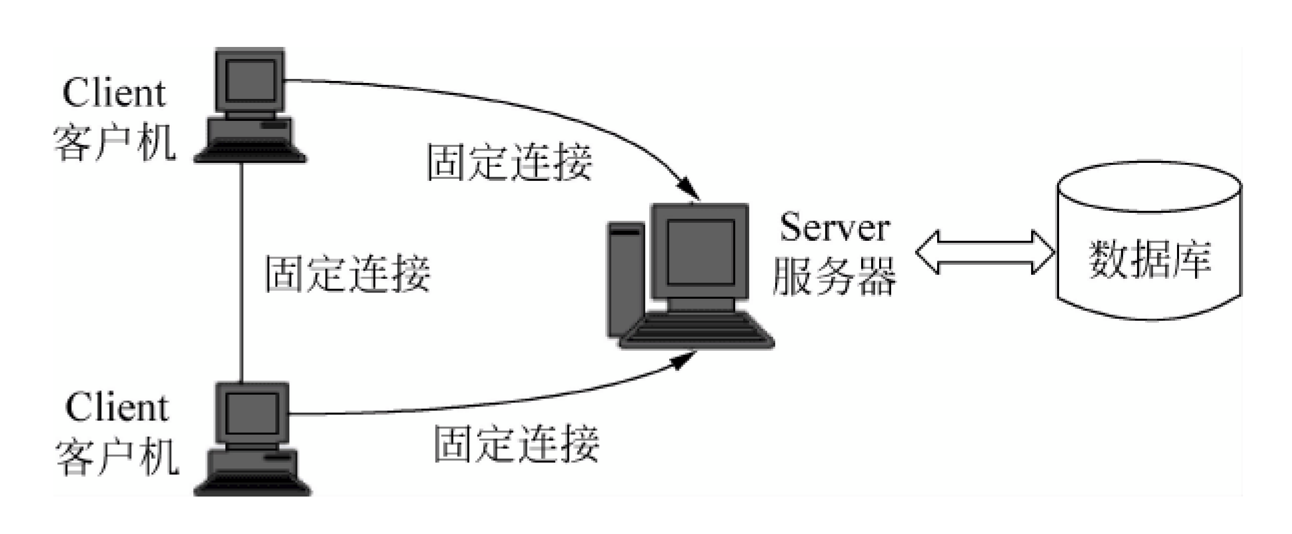


图 1-2 C/S 软件体系结构

1.1.2 B/S 软件体系结构

B/S 是随着 Web 技术发展而逐渐成熟的软件系统体系结构。在这种结构中，所有的应用程序以及数据库系统都安装在服务器(Server)上，客户端只需安装任意一个浏览器(Browser)即可，它是零维护的。用户通过浏览器向服务器发出一个请求(如在地址栏输入一个网址，单击超链接或按钮)，服务器处理该请求后，将结果以 HTML 的形式返回给客户端，如图 1-3 所示。

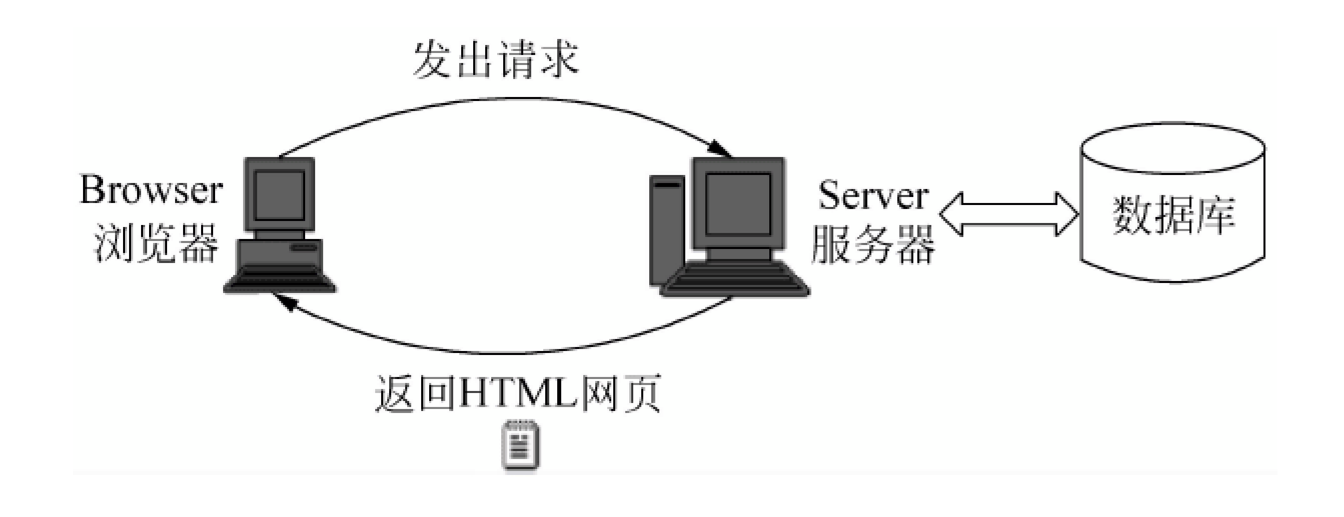


图 1-3 B/S 软件体系结构

B/S 结构采用 Internet/Intranet 技术，用于广域网环境。它可以根据访问量动态地配置 Web 服务器和应用服务器，以支持更多的客户。代码可重用性好，系统扩展维护简单。相对于 C/S 结构而言，B/S 结构具有维护方便、易于升级和扩展、数据集中安全、跨越时空地域限制等特点。

C/S 结构和 B/S 结构的比较如表 1-1 所示。

表 1-1　C/S 结构与 B/S 结构的比较

	C/S 软件体系结构	B/S 软件体系结构
硬件环境	局域网,专门的小范围网络硬件环境,用户固定,用户数量有限	广域网,不必是专门的网络环境,只要是能接入 Internet 的用户均可
系统维护	升级和维护难,成本高	客户端零维护,易于实现系统的无缝升级
软件重用性	单一结构,软件整体性较强,各部分间的耦合性强,可重用性较差	多重结构,各构件相对独立,可重用性较好
平台相关性	客户端和服务器端是平台相关的,多是 Windows 平台	客户端和服务器端是平台无关的
安全性	面向相对固定的用户群,对信息安全的控制能力强	面向不可知的用户群,对信息安全的控制能力相对较弱

1.2　Web 的工作原理

当用户访问 Web 站点时,数据是遵从 HTTP 协议进行传输的。HTTP 即超文本传输协议,是基于 B/S 模式的,使用 TCP 连接在应用层进行可靠的数据传输。HTTP 协议定义了 Web 浏览器和 Web 服务器之间交换数据的过程以及数据本身的格式,是客户机与服务器交互遵守的协议。

1.2.1　HTTP 协议

HTTP 协议的工作原理如图 1-4 所示。它表示基于 HTTP 的信息交换过程,共分为 4 个步骤:建立连接、发送请求信息、返回响应信息、断开连接。首先,客户端的浏览器向服务器的某个端口发出请求,建立与服务器的连接,通常默认端口号为 80。在连接建立后,客户端向服务器发出一个请求(Request)。服务器接收和处理请求后,返回一个响应(Response)页面。最后,客户端与服务器之间的连接断开,通信结束。一般来说,任何一方都可结束连接,但通常是客户端收到所请求的信息后关闭连接。

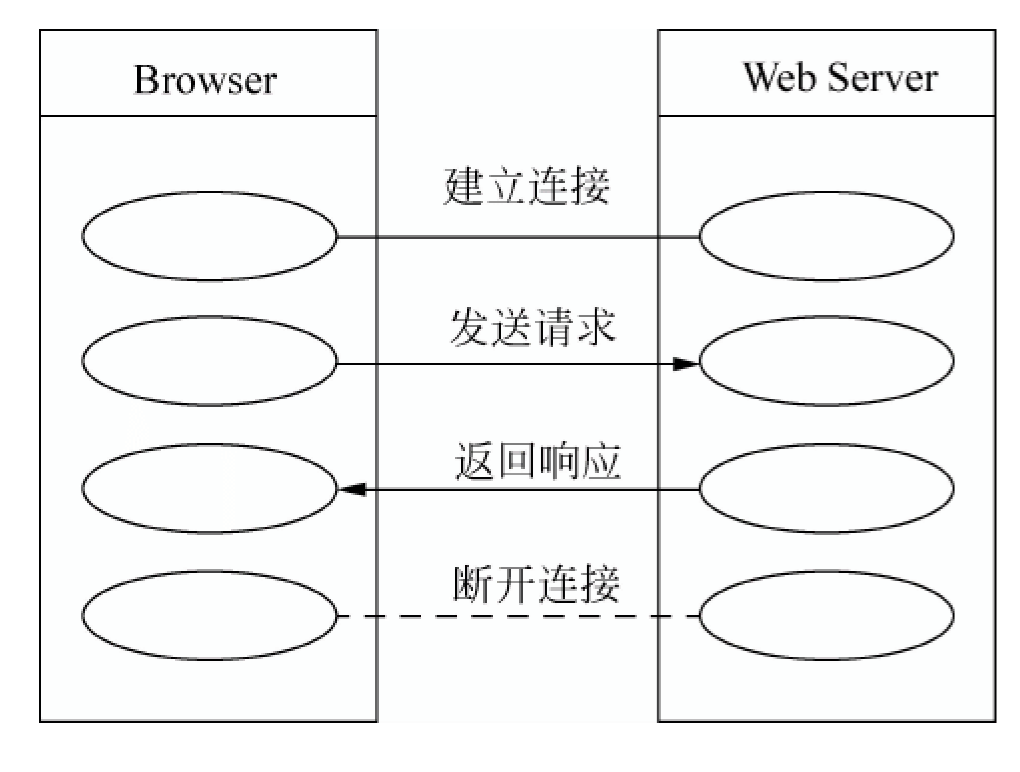


图 1-4　HTTP 协议的工作原理

1.2.2　HTTP 请求和 HTTP 响应

HTTP 协议是一种请求/应答协议,通过客户机和服务器相互发送消息的方式工作。

通常，HTTP 消息包括客户机向服务器的请求消息、服务器向客户机的响应消息。

使用 HTTP 消息头(Header)，可以实现客户机与服务器之间的 HTTP 请求和 HTTP 应答。消息头分为通用头、请求头、响应头和实体头 4 类。

每个消息头包含一个头字段，然后依次是冒号、空格、值、回车和换行符，如 Accept-Language：en-us。消息头字段名是不区分大小写的，但习惯上将每个单词的第一个字母大写。

1. 通用头(General Header)

既能用于请求消息，也能用于响应消息，包括一些与传输内容无关的常用消息头字段。

例如：

```
Cache - Control: no - cache                        ( * )
Connection: close/Keep - Alive                     ( * )
Date: Tue, 11 Jul 2000 18:23:51 GMT
Pragma: no - cache                                 ( * )
Trailer: Date
Transfer - Encoding: chunked                       ( * )
Upgrade: HTTP/2.0, SHTTP/1.3
Via: HTTP/1.1 Proxy1, HTTP/1.1 Proxy2
Warning: any text
```

2. 请求头(Request Header)

一个 HTTP 请求消息包括 3 部分：一个请求行、若干消息头以及消息实体内容。其中一些消息头和实体内容是可选的，消息头和实体内容之间用空行隔开。

HTTP 请求头主要指前两部分，它是客户端向服务器传递的附加信息。

(1) 请求行。这是请求的关键，其格式为：

```
请求方式   请求 URI   HTTP 版本号(回车换行)
```

- 请求方式：指在请求 URI 所指定的资源上实现的方式，可以有 HEAD、GET、POST、OPTIONS、DELETE、TRACE、PUT 等；
- 请求 URI：这里 URI 指标识某一互联网资源的字符串；
- HTTP 版本号。

(2) 消息头。包含浏览器及客户端相关信息，主要有：

- 浏览器类型；
- 支持哪些文档类型；
- 支持哪些字符集、编码、语言；
- 客户机地址。

例如，图 1-5 所示的矩形框就是一个 HTTP 请求消息的内容。第 1 行为请求行，中间的若干行为消息头，空行表示消息头的结束，最后是消息的实体内容。

在图 1-5 中，第 1 行(请求行)表示客户端采用 GET 方式向服务器传送数据，请求的 URI 指向本地文件/books/java.html，使用的 HTTP 协议版本为 1.1。

从第 2 行开始，直到空行前是 HTTP 请求头域的内容。它的常用参数如下：

- Accept：用于指定客户端可以接受的 MIME 类型，如 */* 表示任何类型。

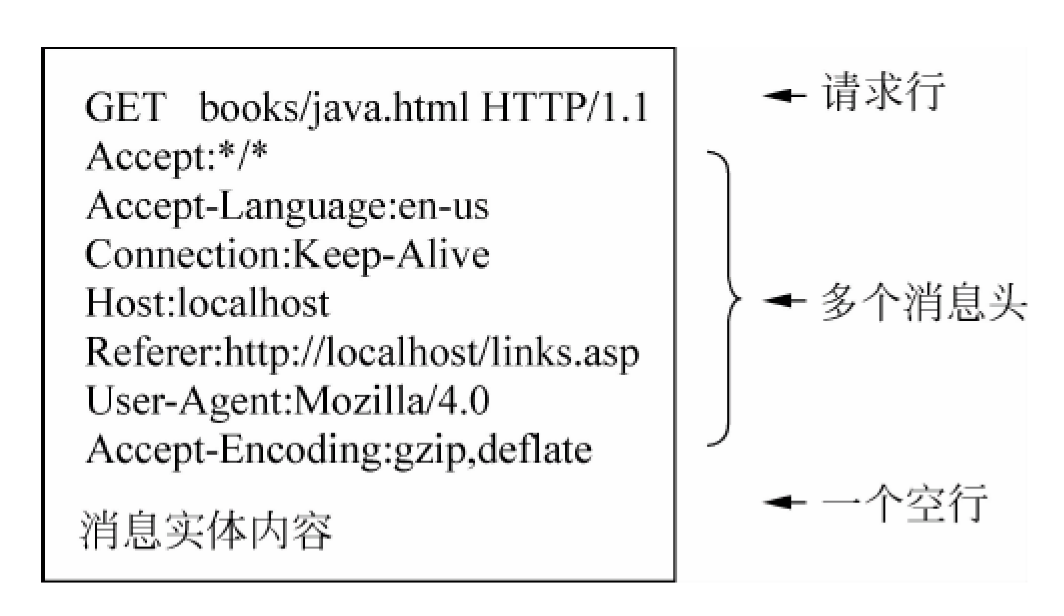


图 1-5 HTTP 请求消息

- Accept-Charset：指定客户端可以使用的字符集，如 UTF-8。
- Accept-Language：指定客户端的接收语言，可以指定多个，如 en-us。
- Accept-Encoding：指定客户端的接收编码，如 gzip、deflate。
- Connection：指示处理完本次请求/响应后，客户端与服务器是否继续保持连接，取值为 Keep-Alive 或 close，默认值为 Keep-Alive。
- Host：指定资源所在的主机和端口号。
- Referer：确定获得请求 URI 的资源地址。
- User-Agent：用户代理，指定初始化请求的客户端程序，如浏览器等。

3. 响应头(Response Header)

同请求消息类似，一个 HTTP 响应消息包括 3 部分：一个状态行、若干消息头以及实体内容。其中，一些消息头和实体内容是可选的，消息头和实体内容之间用空行隔开。

例如，图 1-6 所示为一个 HTTP 响应消息的内容。第 1 行为状态行，中间为消息头，空行表示消息头的结束，最后是消息的实体内容。

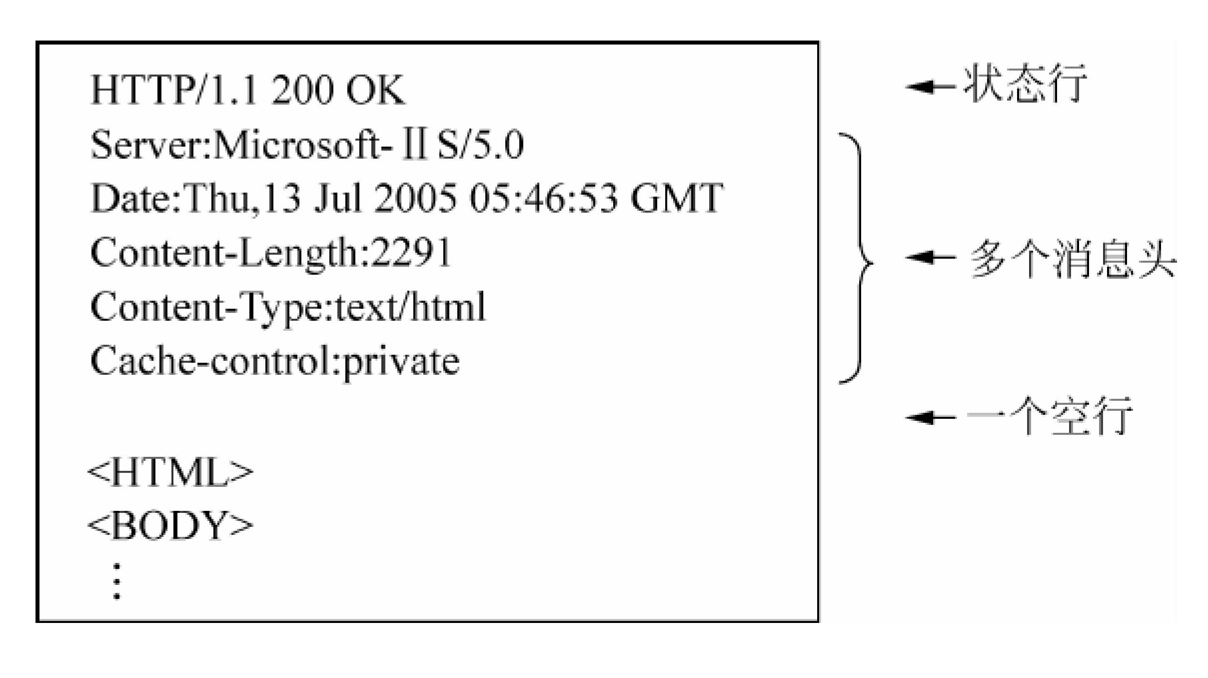


图 1-6 HTTP 响应消息

状态行的格式为：

```
HTTP 版本号  响应状态码  文本描述(回车换行)
```

响应状态码用于表示服务器对请求的各种不同处理结果和状态，是一个 3 位的十进制数。响应状态码可归为 5 种类别，分别如下：

- 100～199：表示成功接收请求，要求客户端继续提交下一次请求。
- 200～299：表示成功接收请求并已完成整个处理过程。

- 300～399：为完成请求，客户需进一步细化请求。
- 400～499：客户端的请求有错误。
- 500～599：服务器端出现了错误。

例如，典型的状态码如下：

- 200(OK)表示服务器已成功处理了请求，返回的是正常请求结果。
- 201(Created)请求已经实现，并且服务器创建了新的资源。
- 206(Partial Content)：服务器已经成功处理了部分 CET 请求。
- 302/307(Redirect Temporarily)：指出被请求的文档已被临时移动到别处，此文档的新的 URL 在 Location 响应头中给出。
- 304(Not Modified)：表示客户机缓存的版本是最新的，客户机应该继续使用它。
- 400(Bad Request)：语义有误，当前请求无法被服务器理解。
- 401(Unauthorized)：当前请求需要用户验证。
- 403(Forbidden)：服务器拒绝执行当前请求。
- 404(Not Found)：服务器上不存在客户机所请求的资源。
- 500(Internal Server Error)：服务器端的程序发生错误，导致无法完成对请求的处理。

4. 实体头(Entity Header)

请求消息和响应消息都可以包含实体信息，实体信息一般由实体头和实体内容组成。实体头描述了实体内容的属性，包括实体信息类型、长度、压缩方法、最后一次修改时间、数据有效期等。

例如：

```
Allow: GET,POST
Content-Encoding: gzip                                      (*)
Content-Language: zh-cn                                     (*)
Content-Length: 80                                          (*)
Content-Location: http://www.it315.org/java_cn.html
Content-Range: bytes 2543-4532/7898                         (*)
Content-Type: text/html; charset=GB2312                     (*)
Expires: Tue, 11 Jul 2000 18:23:51 GMT                      (*)
Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT                (*)
```

1.3 Web 程序设计技术

1.3.1 静态网页和动态网页

在 Web 服务器返回客户端的页面中，有静态网页和动态网页之分。它们的区别在于服务器对它的处理方式不同，了解这种区别对于理解 Web 程序设计的概念至关重要。

1. 静态网页

网站设计中，纯粹 HTML 格式的网页被称为"静态网页"，通常是以 .htm 或 .html 为后缀的 HTML 文件。它可以包含 HTML 标记、客户端脚本，但不包含任何服务器端脚本。静态网页也可以出现动态的效果，如 gif 动画、Flash 等，但这与动态网页是不同的概念。

静态网页的主要特点如下：

- 每个静态网页都有一个固定的 URL 地址；
- 静态网页的内容是原封不动被传递的，如果要修改网页内容，必须修改 HTML 源代码；
- 静态网页没有数据库的支持，不支持客户端和服务器端的交互。

静态网页的处理流程如图 1-7 所示。

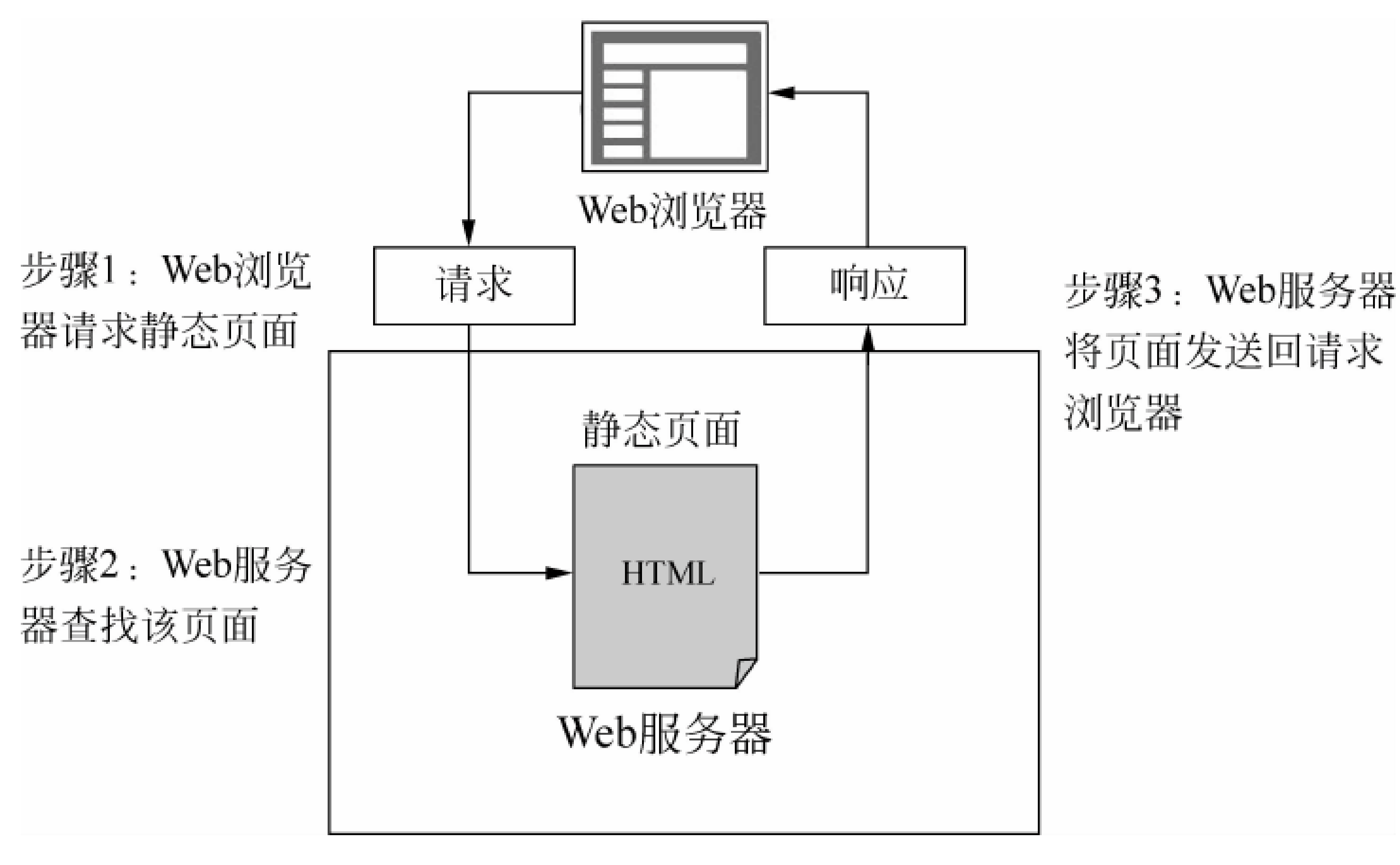


图 1-7　静态网页的处理流程

2. 动态网页

所谓"动态"，就是指服务器根据用户的不同请求做出不同的响应。动态网页文件中含有 HTML 标记和程序代码，后缀则随程序语言的不同而不同，如. asp、. aspx、. php、. jsp 等，大都需要数据库支持。

动态网页的主要特点概括如下：

- 动态网页是由服务器执行相应的脚本程序后，动态生成的 HTML 文件；
- 不同的请求及访问数据的变化会生成不同的 HTML 代码，网页内容随时更新；
- 具有数据库访问功能，支持客户端和服务器端的交互。

动态网页的处理流程如图 1-8 所示。

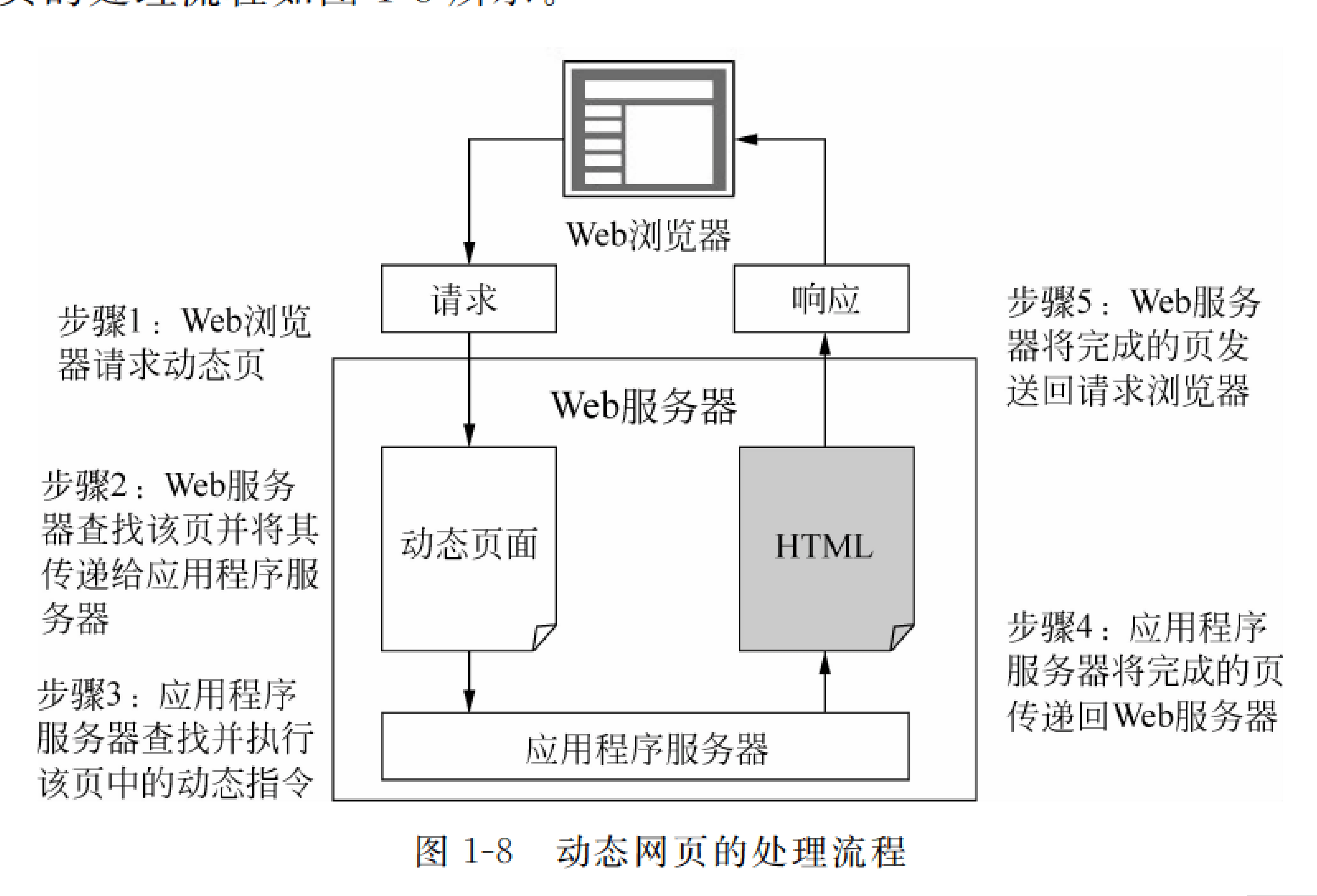


图 1-8　动态网页的处理流程

注意：Web服务器返回给客户端的页面都是纯粹的HTML代码，最后由浏览器解释执行这些代码，并将结果显示在窗口中。

1.3.2 客户端和服务器端脚本编程

脚本(Scripts)是指嵌入到Web页面中的程序代码，所使用的编程语言称为脚本语言。按照执行方式和位置的不同，脚本分为客户端脚本和服务器端脚本。客户端脚本在客户机上被Web浏览器执行，服务器端脚本在服务器上被Web应用服务器执行。

脚本编程的基本特点是，由HTML构造页面模板，中间混合脚本代码，由语言解释引擎解释并运行脚本，产生页面的内容插入到页面的模板中。客户端脚本的解释器位于Web浏览器中，服务器脚本的解释器则位于Web服务器中。

静态网页只能包含客户端脚本，动态网页则可以包含客户端脚本和服务器端脚本。因此，静态网页中的任何脚本都不能在服务器上运行，而动态网页中的某些脚本只能在Web服务器上运行。

1. 客户端脚本编程(Client-side Scripting)

常见的客户端脚本语言有VBScript、JScript、JavaScript等。Microsoft公司开发了VBScript和JScript，Netscape公司开发了JavaScript。它们都是解释型、基于对象的脚本语言。它们有着相同的工作机制：将脚本嵌入到Web页面中，并随着HTML文件一起传送到客户端，由浏览器解释执行。在脚本执行期间无须与服务器交互，可以对用户的操作直接做出响应。

用客户端脚本语言编写的程序通常用标记<script>和</script>界定，可以放到HTML的<head>或<body>部分。

1) VBScript

VBScript是Visual Basic家族的成员，将灵活的脚本应用于更广泛的领域，包括IE浏览器的客户端脚本和Internet Information Server中的服务器端脚本。不过Netscape公司的Navigator浏览器不支持VBScript脚本，因此最好不要在客户端使用VBScript语言，在服务器端则不必考虑浏览器的支持问题。

2) JScript

JScript只能用于IE浏览器。最初的JScript和JavaScript差异过大，Web程序员不得不为两种浏览器编写两种脚本，于是诞生了ECMAScript，它是一种国际标准化的JavaScript版本。现在的主流浏览器都支持这种版本。现在的JScript和JavaScript非常类似，在使用上基本不用担心其不同。

3) JavaScript

JavaScript的正式名称是ECMAScript。JavaScript介于Java与HTML之间，是一种基于对象和事件驱动并具有安全性能的脚本语言。

JavaScript的特点：不需要Java编译器，由浏览器逐行地解释执行；JavaScript具有跨平台性，依赖于浏览器本身，与操作环境无关，只要是能支持JavaScript的浏览器就可以正确执行；JavaScript使网页变得生动；JavaScript可以使有规律重复的HTML程序段简化，减少下载时间；JavaScript能及时响应用户的操作，对提交表单做即时的检查，大大减少了服务器的开销。

2. 服务器端脚本编程(Server-side Scripting)

动态网页的生成需要执行服务器端的脚本程序,这些程序就是由“服务器端的脚本语言”编写而成的。常见的脚本语言有 ASP(.NET)、PHP 和 JSP 等。它们具有的共同点是,脚本都运行于服务器端,能够动态地生成网页,脚本运行不受客户端浏览器限制;脚本程序都是将脚本语言嵌入到 HTML 文档中,执行后返回给客户端的是 HTML 代码。

下面对它们各自的特点做一个简单的介绍。

1) PHP(Hyper Text Preprocessor)

PHP 为 PHP 网络小组所开发,是免费的开放源代码产品。Apache 和 MYSql 也同样是免费开源,Apache、PHP 和 MYSql 搭配使用,可以快速地搭建一套动态网站系统,其执行效率比 IIS+ASP+Access 要高,而后者的使用还必须另外付钱给微软公司。

PHP 是一种嵌入 HTML 页面中的脚本语言,类似于 C 语言、Java 和 Perl,语法简单,是一种解释型语言。它可以跨 UNIX、Linux 、Windows 平台使用,具有平台无关性。PHP 非常利于快速开发各种功能不同的定制网站,运行成本低。不足之处是运行环境的安装、配置比较复杂,同时由于 PHP 内部结构上的缺陷,使得 PHP 在复杂的大型项目上的开发和维护都比较困难。

2) JSP(Java Server Pages)

JSP 是由 Sun 公司推出的,是基于 Java Servlet 以及整个 Java 体系的 Web 开发技术。在 HTML 文档中嵌入 Java 程序片段(Scriptlet)和 JSP 标记,就形成 JSP 文件。JSP 脚本在服务器端运行,可以跨 UNIX/Linux/Windows 平台使用,具有平台无关性。JSP 代码需编译成 Servlet 并由 Java 虚拟机执行,编译操作仅在对 JSP 页面的第一次请求时发生。

JSP 最大的好处就是开发效率高,可以使用 JavaBeans 或者 EJB(Enterprise JavaBeans)来执行应用程序所要求的复杂处理,但这种网站架构因为其业务规则代码与页面代码混为一体,不利于维护,因此不适应大型应用的要求,取而代之的是基于 MVC 的 Web 架构。通过 MVC 的 Web 架构,可以弱化各个部分的耦合关系,并将业务逻辑处理与页面以及数据分离开来,这样当某个模块的代码改变时,并不影响其他模块的正常运行。因此,不少国外的大型企业系统和商务系统都使用基于 Java 的 MVC 架构,能够支持高度复杂的基于 Web 的大型应用。

3) ASP(Active Server Pages)

ASP 是 Microsoft 开发的 Windows IIS(Internet Information Services)系统自带的脚本语言,是一种简单、方便的脚本编程工具。ASP 文件可以包含 HTML 标记、普通文本、脚本命令等,它是由浏览器解释执行的。ASP 是平台相关的,只能在 Windows 平台运行,数据库最好为 Microsoft SQL Server 或 Access。

ASP 的优点是简单易学,缺点是运行速度比 HTML 的运行速度慢,无法实现跨平台应用。ASP 易于维护,很适合小型网站应用。

4) ASP.NET(Active Server Pages .NET)

ASP.NET 又叫 ASP+,是 Microsoft 的 Active Server Pages 的新版本,但它不是 ASP 的简单升级,而是 Microsoft 推出的新一代脚本语言。ASP.NET 是.NET 框架体系的一部分,它建立在公共语言运行库上,可用于在服务器上生成功能强大的 Web 应用程序,为 Web 站点创建动态的、交互的 HTML 页面。ASP.NET 的主要特点如下:

- ASP.NET 采用编译后运行的方式，执行效率大幅提高。它将程序在第一次运行时编译成 DLL 文件，以后直接执行 DLL 文件，这样速度就变得非常快。
- 世界级的工具支持。ASP.NET 程序采用 Microsoft 的 Visual Studio.NET 环境进行开发，支持 WYSIWYG（所见即所得）、拖放控件和自动部署等功能。
- 程序结构清晰。ASP.NET 将程序代码和 HTML 标记分开，使程序设计相对简化，结构更为清晰，但同时也加重了页面设计的难度。
- 移植方便。ASP.NET 可以向目标服务器直接复制组件，当需要更新时，重新复制一个即可。

1.4　习题与上机练习

1. 填空、选择题

（1）不使用 IP 地址，可以使用________访问本机上的默认 Web 主页。

（2）下列不是动态网页文件后缀的是________。

A. asp　　B. php　　C. jsp　　D. html

（3）下列________方式无法正常浏览 ASPX 文件的运行结果？

A. http://localhost/test.aspx　　B. http://127.0.0.1/test.aspx

C. http://IP 地址/test.aspx　　D. 在浏览器地址栏直接输入 test.aspx

（4）对动态网页的正确理解是________。

A. 动态网页就是网页里有动态图片的网页

B. 动态网页就是网页里有动画的网页

C. 动态网页就是网页内容动态生成的网页

D. 动态网页就是指所有包含动态图片和动画的网页

（5）在客户端网页脚本语言中最为通用的是________。

A. JavaScript　　B. Visual Basic　　C. Perl　　D. ASP

2. 简答题

（1）C/S 结构与 B/S 结构有什么区别？

（2）静态网页和动态网页有什么区别？

（3）客户端脚本语言和服务器端脚本语言有什么区别？

第 2 章　HTML 与 XML

Web 应用程序主要通过 HTML 和 XML 来表达信息。

HTML 定义了 Web 网页的文档结构，用固有的标记来描述和显示网页内容。XML 定义了 Web 信息本身的数据形式和结构，不能描述网页的外观和内容。HTML 和 XML 在 Web 应用开发中有着密切的关系。

2.1　使用 HTML 设计网页

超文本标记语言（Hypertext Marked Language，HTML）是 Web 页面的标记性语言，通过一定的格式标记文本及图像等元素，使之在浏览器中显示出不同内容和风格的网页。

HTML 文档通过浏览器在 WWW 上发布，并独立于各种操作系统平台。一个 Web 页面就是一个 HTML 文档。Web 页面也就是通常所说的网页，在本书中不作区分。

HTML 文件本身是一种文本文件，通过在文件中添加各种标记，告诉浏览器如何显示其中的内容（如文字如何显示、画面如何安排等）。浏览器按顺序解释执行 HTML 文件，对于书写出错的标记既不报错，也不停止执行。需要注意的是，不同的浏览器对于同一标记可能有不同的解释，也就可能有不同的显示效果。

2.1.1　HTML 文档的基本结构

HTML 文档分为文档头和文档体两部分，文档头对当前文档进行了一些必要的定义，文档体才是要显示的各种信息。

下面是一个最基本的 HTML 文档结构：

```
<html>                                        //文档开始
    <head>                                    //文档头开始
        <title></title>                       //文档标题
        <meta content = "text/html">          //属性标记
    </head>                                   //文档头结束
    <body>                                    //文档体开始
    </body>                                   //文档体结束
</html>                                       //文档结束
```

1. 文档开始和结束标记

HTML 文档的开始标记是<html>，在文档结束处的标记是</html>。所有其他标记都必须置于<html>和</html>标记之间，否则浏览器将忽略不在其间的标记。

2. 文档头标记

HTML 文档的头标记是<head>和</head>，用于初始化文档信息。

头标记<head>中包括标题标记<title>和属性标记<meta>。

<title>和</title>之间的内容是指当前网页的标题，通常在 64 个字符以内。

<meta>标记是单标记，用于指定 HTML 文档的特殊属性。例如，http-equiv 属性相当于 http 文件头；name 属性用于描述网页；content 属性是 name 属性所赋的值。

例如：

```
<meta http-equiv = "refresh" content = "10";url = "http://www.sina.com.cn">
```

说明当前 HTML 文档 10 秒后自动刷新，并跳转到 url 指向的另一个主页。

又如：

```
<meta http - equiv = "Content - Type" Content = "text/html";charset = GB2312 >
```

说明页面的文件类型(HTML 文档)以及所用字符(GB2312 字符集)。

3. 文档主体标记

HTML 文档的主体标记是<body>和</body>，是 HTML 文档最核心的部分，Web 页面的主要内容都放在这里。它的语法格式为：

```
<body [background = # | bgcolor = # | text = # | link = # | alink = # | vlink = # | leftmargin =
# | topmargin = # ]>
    …
</body >
```

2.1.2 HTML 文档的主要标记

2.1.2.1 文字和段落标记

1. 标题字体标记 <Hn>

这里的标题是指 HTML 页面中文本的标题，而不是指网页标题。

标题元素有 6 种，分别为 h1、h2、h3、h4、h5、h6，用于表示文章中的各种题目。标题号越小，字体越大，即 h1 是特大字体。

例如：

```
< h1 > 标题文字</h1 >
```

**2. 字体标记 **

使用和标记可以格式化文本，使文字呈现出不同的字体、字号、颜色等。

使用格式为：

```
< font size = 字号大小 face = 字体 color = 颜色> 被设置的文字 </font >
```

说明：

(1) size 属性用来设置字号的大小，其取值范围从小到大依次为 1～7。

(2) face 属性用来设置不同的字体，如宋体、楷体_GB2312、Times New Roman 等。

(3) color 属性用来设置文字颜色。其值可以是 red，blue，＃000000 等。

颜色名称和十六进制数码的对应关系如表 2-1 所示。

表 2-1　颜色名称和对应的十六进制数码

Black＝"＃000000"	Green＝"＃008000"
Silver＝"＃C0C0C0"	Lime＝"＃00FF00"
Gray＝"＃808080"	Olive＝"＃808000"
White＝"＃FFFFFF"	Yellow＝"＃FFFF00"
Maroon＝"＃800000"	Navy＝"＃000080"
Red＝"＃FF0000"	Blue＝"＃0000FF"
Purple＝"＃800080"	Teal＝"＃008080"
Fuchsia＝"＃FF00FF"	Aqua＝"＃00FFFF"

3. 字体样式标记

通过字体样式标记可以给文本设置一些特殊格式，如粗体、斜体、下划线、上下标等。其标签可以分为物理样式和逻辑样式两类，分别如表 2-2 和表 2-3 所示。

表 2-2　物理样式标记

属　性	说　明	属　性	说　明
<B>…</B>	黑体字	[…]	上标
<I>…</I>	斜体字	_…	下标
<U>…</U>	加下划线	<SMALL>…</SMALL>	小字体
<S>…</S>	加删除线	<BIG>…</BIG>	大字体

表 2-3　逻辑样式标记

属　性	说　明
<EM>…</EM>	输出需要强调的文本(通常是斜体加黑体)
<STRONG>…</STRONG>	输出加重文本(通常也是斜体加黑体)
<CITF>…</CITF>	输出引用方式的字体，通常是斜体
<DEL>…</DEL>	为文本加上删除线

4. 文字布局标记

将文档划分为段落，可以通过分段标记、换行标记、标题标记或插入水平线来实现。

1) 分段标记 <p>

分段标记<p>和</p>定义了一个段落，并在段与段之间加一个空行，使后续行隔行显示。它的常用属性是 align，表示对齐方式，其取值包括 right(右对齐)、left(左对齐)、justify(两端对齐)、center(居中对齐)。

2) 换行标记

换行标记
强行规定了当前行的中断，使后续内容在下一行显示，两行间不隔行。

例如，下述两行 HTML 代码就得到不同的效果：

```
你好吗?<p>很好。
你好吗?<br>很好。
```

3) DIV 标记

DIV 标记用于为文档分节，以便在文档的不同部分应用不同的段落格式。单独的 DIV 标记不能完成任何工作，必须与 align 属性联合使用。位于 DIV 标记符中的多段文本将被认为是一个节，可为它们设置一致的对齐格式。

DIV 标记的格式为：

```
<DIV align = left|center|right > 文本或图像 </DIV>
```

4) 水平线标记 <hr>

水平线标记<hr>用于在网页中添加一条水平线，将不同的信息内容分开。其格式为：

```
<hr align = 对齐方式 size = 粗细 width = 长度 noshade>
```

其中，属性 align 指示对齐方式；size 指定粗细，单位为像素；width 指定线的长度，其值为像素或百分比；noshade 表示一条无阴影的实线，若省略 noshade，则显示带阴影的三维实线。

2.1.2.2 超链接

所谓超链接(Hyperlink)，就是当单击网页上某些内容时，可以打开另一个网页内容。超链接的组织体现了 Web 站点内部或不同站点之间的页面存储的逻辑关系。

超链接的语法格式为：

```
<a href = URL 地址 title = 标题 target = 窗口名称> 链接文本 </a>
```

说明：href 属性表示链接所指向的 URL 地址。title 属性指定指向链接时所显示的标题文字。target 属性指定链接对象的显示位置，即打开链接的目标窗口，默认是原窗口。

针对链接对象的地址不同，可以分为内部链接、外部链接和书签链接。

1. 内部链接

内部链接是指链接到本地主机的文件。

例如：

```
<a href = "1.htm"> 请单击查看 1.htm 中的内容 </a>
```

2. 外部链接

外部链接是指链接到非本地主机上的文件，可以是其他主机或任意站点上的某个文件。

例如：

```
<a href = "http://www.xatu.edu.cn/">链接到主页</a>
<a href = "telnet://bbs.xatu.edu.cn">远程登录</a>
```

3. 书签链接

书签链接是指链接到同一页面中的某个特定位置，相当于在页面的某个地方做上书签，

需要时通过书签链接可以快速找到该部分。

例如，在某个页面中，使用 name 属性定义一个名为 first 的书签。

```
<a name = "first"> 第一章 </a>
```

在同一页面的另外一个位置使用 href 属性来指向它。

```
<a href = "#first"> 指向第一章 </a>
```

当用户单击超链接"指向第一章"时，页面就会跳转到 first 书签所在位置。注意，引用书签时要加上"#"号。

书签链接的基本格式如下：

(1) 在同一页面中，要使用书签名：

```
<a href = "#书签名">  超链接标题名称  </a>
```

(2) 在不同页面之间，要使用链接的 URL 地址：

```
<a href = "URL 地址#书签名"> 超链接标题名称  </a>
```

2.1.2.3 列表

列表(LIST)通常用于列举相关的信息条目，提供一种有组织的、易于浏览的阅览格式，使得文本的条理更清晰。常见的列表有三种：无序列表、有序列表和定义列表。

1. 无序列表<ul>

无序列表中每一个表项的前面是项目符号，使用<ul>标记和<li>表项标记。其格式为：

```
<ul type = 符号类型>
    <li> 列表项 1 </li>
    <li> 列表项 2 </li>
         ⋮
</ul>
```

标记有一个 type 属性，用于指定列表项前面的项目符号，且必须小写。取值为 disc 表示实心圆●(默认值)；circle 表示空心圆○；square 表示小方块■。

在同一个列表中，
也可以省略，下一个<li>的出现，就表明上一个列表项的结束。

2. 有序列表<ol>

有序列表使用标签
和
表示，使内容按顺序编号。每插入或删除一个列表项，编号会自动调整。其格式如下：

```
<ol type = 符号类型 start = 起始编号>
    <li> 列表项 1 </li>
    <li> 列表项 2 </li>
     ⋮
</ol>
```

标记有两个属性：start 属性和 type 属性。start 表示起始编号，默认为 1。type 属性设置列表项前面的数字序列样式，其取值如表 2-4 所示。

表 2-4　有序列表的 type 属性取值

取　值	说　明
type＝1	列表项用数字编号(1,2,3…)，默认值
type＝A	列表项用大写字母编号(A,B,C…)
type＝a	列表项用小写字母编号(a,b,c…)
type＝I	列表项用大写罗马数字编号(Ⅰ，Ⅱ，Ⅲ…)
type＝i	列表项用小写罗马数字编号(i,ii,iii…)

3. 定义列表<dl>

定义列表用于给每一个列表项加上一段说明性文字，说明性文字独立于列表项，另起一行显示。

定义列表以<dl>和</dl>作为起止标记，列表项用<dt>引导，说明性文字用<dd>来引导。<dt>与<dd>不需要结尾标记。其格式如下：

```
<dl>
<dt>第一项 <dd>叙述第一项的定义
<dt>第二项 <dd>叙述第二项的定义
⋮
</dl>
```

例 2-1　列表标记的使用示例。

```
<html>
<head><title>列表标记示例</title></head>
<body>
<ol type=1>
<li><u>无序列表</u>
    <ul type=circle>
        <li>Photoshop
        <li>Illustrator
        <li>CorelDraw
    </ul>
<li><u>有序列表</u>
    <ol type=a>
        <li>Photoshop
        <li>Illustrator
        <li>CorelDraw
    </ol>
<li><u>定义列表</u>
    <dl>
        <dt>Photoshop<dd>Adobe 公司的图像处理软件
        <dt>Illustrator<dd>Adobe 公司的矢量绘图软件
        <dt>CorelDraw<dd>Corel 公司的图形图像软件
    </dl>
</ol>
<body>
</html>
```

上述代码定义了一个嵌套的列表。第一级是以数字为标号的有序列表，里面分别嵌套了无序列表、有序列表和定义列表作为第二级，并使用了<u>标记加注了下划线。运行结果如图 2-1 所示。

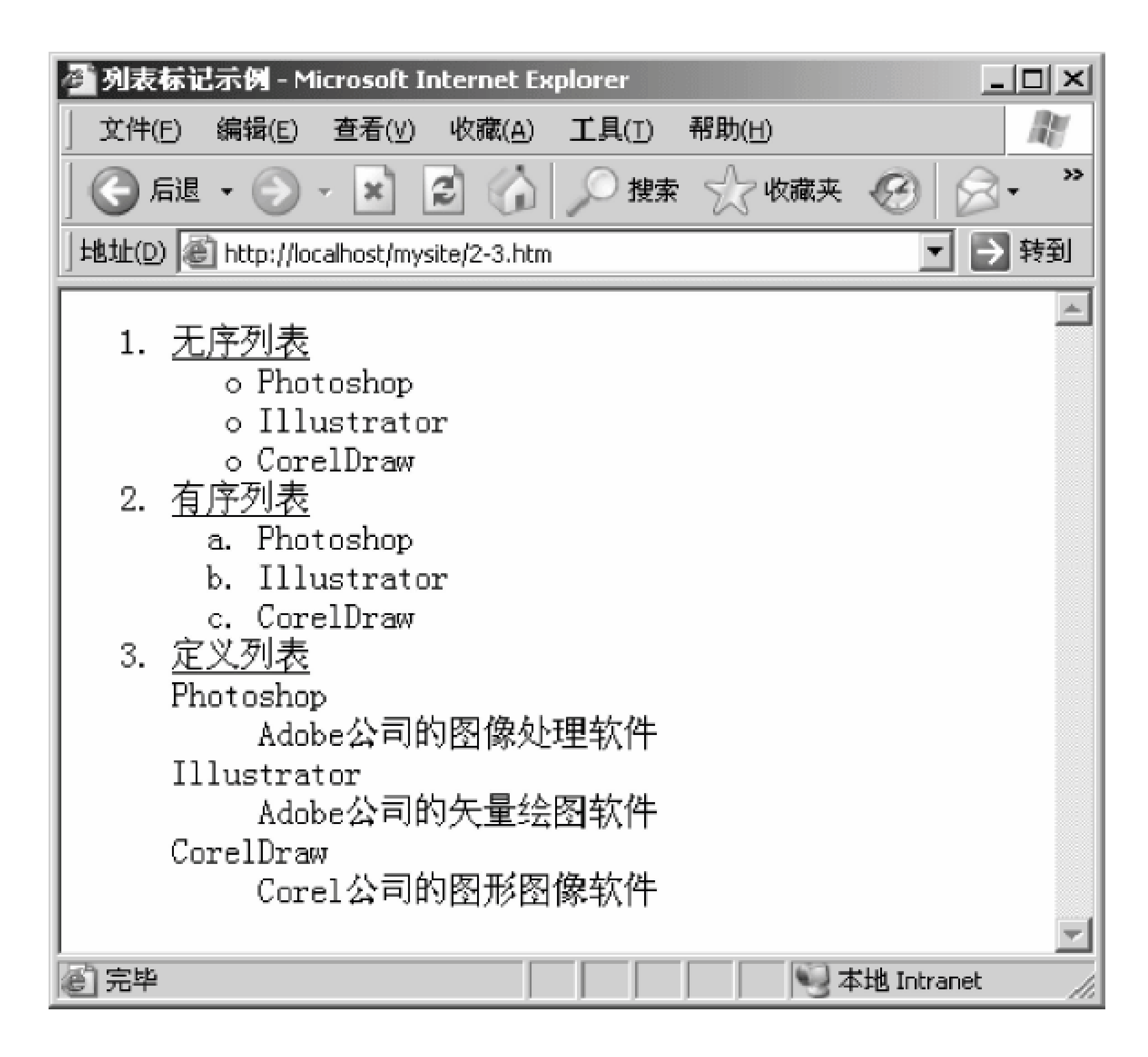


图 2-1　列表标记示例

2.1.2.4　表格

表格(Table)将文本和图片按行、列编排，有利于表达信息。在主页中往往使用表格建立主页的框架，使整个页面更规则地放置图片和空白，以使条目清晰。用表格的语法格式如下：

```
<table align="left|center|right" border="n" width="x|x%" height="y|y%">
    <tr><th>表头 1<th>表头 2…<th>表头 n
    <tr><td>表元 1<td>表元 2…<td>表元 n
     ⋮
    <tr><td>表元 1<td>表元 2…<td>表元 n
</table>
```

1. <table>标记

表格的标记为<table>，行的标记为<tr>，表头的标记为<th>，表项的标记为<td>。其中，<tr>可以是单标记，一行的结束可以是新行的开始。有关说明如表 2-5 所示。

表 2-5　表格标记

标　　签	说　　明
<table>…</table>	定义一个表格的开始和结束
<tr>…</tr>	定义表行，一组行标签内可以建立多组由<td>或<th>所定义的单元格
<th>…</th>	定义表头，表头中的文字将以粗体显示，<th>必须放在<tr>标签内
<td>…</td>	定义表元，一组<td>标签将建立一个单元格，<td>必须放在<tr>标签内

表格的整体外观由<table>标记的属性决定。创建表格时，可以通过<table>的属性对表格的格式进行设置，其取值如表 2-6 所示。

表 2-6　<table>标记的属性

属　　性	说　　明
align	指定表格的对齐方式，取值可以是 left、center 或 right
background	指定用作表格背景图片的 URL 地址
bgcolor	指定表格的背景颜色
border	指定表格边框的宽度，以像素为单位。如果省略该属性，则默认值为 0
bordercolor	指定表格边框颜色，应与 border 属性一起使用
bordercolordark	行的暗边框颜色，应与 border 属性一起使用
bordercolorlight	指定 3D 边框的高亮显示颜色，应与 border 属性一起使用
cellpadding	指定单元格内数据与单元格边框之间的间距，以像素为单位
cellspacing	指定单元格之间的间距，以像素为单位
width	指定表格的宽度，以像素或百分比为单位

2. <caption>标记

用<caption>标记给表格加标题，其格式为：

```
<caption align = "left|right|top|bottom" valign = "top|bottom"> 标题 </caption>
```

3. 跨多行、多列的单元格

跨多行、多列的单元格使用 colspan 和 rowspan 属性进行定义。

1）跨多列的单元格

```
<th colspan = n> 表项 </th> 或 <td colspan = n> 表项 </td>
```

其中，n 表示合并的列数。

2）跨多行的单元格

```
<th rowspan = m> 表项 </th> 或 <td rowspan = m> 表项 </td>
```

其中，m 表示合并的行数。

例 2-2　跨多行、多列的单元格。

```
<HTML>
<HEAD><TITLE>单元格跨多行、多列</TITLE></HEAD>
<BODY>
<TABLE align = center border = 5 >
<CAPTION><FONT size = 5 ><B>学生成绩表</B></FONT></CAPTION>
<TR><TH rowspan = 2 >学号<TH rowspan = 2 >姓名<TH colspan = 3 >成绩
<TR><TH > C 语言设计<TH >数据结构<TH >总分数
<TR><TD> 0001 <TD>张  林<TD> 80 <TD> 93 <TD> 173
<TR><TD> 0002 <TD>刘亚玲<TD> 90 <TD> 9100 <TD> 180
<TR><TD> 0003 <TD>赵  丽<TD> 72 <TD> 88 <TD> 160
<TR><TD> 0004 <TD>李  进<TD> 85 <TD> 85 <TD> 170
```

```
<TR><TD>0005<TD>李 萍<TD>90<TD>60<TD>150
</TABLE>
</BODY>
</HTML>
```

显示结果如图 2-2 所示。



图 2-2 跨多行、多列的单元格

2.1.2.5 设计表单

表单是实现服务器与用户之间交互信息的主要方式。它最直接的作用就是从客户端浏览器收集信息，并指明一个处理信息的方法。

1. 表单标记

表单是用户和 Web 应用程序进行交互的界面。用户在表单中填写完信息后，做提交表单的操作，表单内容就从客户端传送给 Web 服务器，服务器上的应用程序处理后，将结果传送回客户端。

表单用<form>和</form>标记创建。其语法格式如下：

```
<form name=表单名 action=处理程序名 method=方式 target=目标窗口>
    <input type=# name=## value=… />
    …
</form>
```

说明：

(1) name 属性：指定表单名称。

(2) action 属性：指定服务器端处理程序的 URL 地址。

(3) method 属性：定义客户端提交数据的方式，取值有 GET 和 POST。

GET 方式是将表单数据附加在 action 指定的 URL 地址之后，并在 URL 与数据之间加上一个分隔符“?”，各数据项之间用“&”进行分隔。

例如：

```
http://www.website.net/example.aspx?txtID=007&txtName=James
```

由于浏览器地址栏长度的限制,GET 方法一次最多只能提交 256 个字符的数据。

POST 方法与 GET 方法不同,它把表单数据作为一个独立的数据块直接传递给服务器,传送的数据量要比 GET 方式大得多,且不受长度限制。

(4) target 属性:用来指定目标窗口。其值有_blank(空白窗口)、_parent(父级窗口)、_self(当前窗口)和_top(顶层窗口)。默认为当前窗口。

2. 表单域标记

常见的表单域有以下几种:文本域、按钮域、选择域、菜单和列表域等。

1) 文本域

文本域根据输入方式的不同,可分为单行文本域、密码文本域和多行文本域。

(1) 单行文本域:输入文本以单行显示,其类型为 type=TEXT。

```
<input type="TEXT" name=名称 value=内容 maxlength=最大字符数 size=宽度>
```

其中,name 表示文本域的名称;value 表示默认值;maxlength 表示最大可输入字符数;size 表示文本域的宽度(以字符为单位)。

(2) 密码文本域:输入到文本域中的文字均以圆点的形式显示。方法为:

```
<input type="PASSWORD" name=名称 maxlength=最大字符数 size=宽度>
```

(3) 多行文本域:如果输入的文本较多,可使用<textarea>和</textarea>标签创建一个能够输入多行的文本框。

在多行文本框中输入数据时,如果一行的内容过多,或行数过多,则会自动加上水平和垂直滚动条。其语法格式为:

```
<textarea name=名称 value=初始值 rows=行数 cols=列数> </textarea>
```

2) 按钮域

使用按钮可以提交表单或重置表单。按钮分为三类:

(1) 提交按钮(type=submit):单击提交按钮,可以实现表单内容的提交。

```
<input type="submit" name=名称 value="提交">
```

(2) 重置按钮(type=reset):单击重置按钮,可以清除表单中已经输入的内容。

```
<input type="reset" name=名称 value="重置" >
```

(3) 普通按钮(type=button):使用普通按钮可以通过调用函数完成其他操作。

```
<input type="button" name=名称 value=文本>
```

3) 选择域

选择域根据输入方式不同,分为单选域和复选域两类。

(1) 单选域(type=radio)用来在多个选项中选取一项。格式如下:

```
<input type="radio" name=名称 value =单选域的取值 checked>
```

其中，checked 表示默认选中的项；value 表示选中项传送到服务器的值。

(2) 复选域(type=checkbox)用于进行多项选择。基本格式为：

```
<input type="checkbox" name=名称 value=取值 checked>
```

4) 菜单和列表域

假如在表单中需要添加很多内容，可以使用<select>标签来实现。格式如下：

```
<select name=菜单名称 size=选项个数 multiple>
    <option value=选项值 1 selected> 显示内容 1 </option>
    <option value=选项值 2> 显示内容 2 </option>
    ⋮
    <option value=选项值 n> 显示内容 n </option>
</select>
```

说明：

- size 表示选项个数(默认是 1)。如果大于 1，则为多选列表；等于 1 则表示单选菜单(即下拉菜单)。
- multiple 表示可以复选，即选择多个选项。
- value 表示选项的值，通过检测该菜单的 value 值，可以知道用户选择了哪一项。
- selected 表示当前选项在初始状态被选中。

例 2-3 表单的综合应用。

```
<html>
<head><TITLE>表单的综合应用</TITLE></head>
<body>
<form action="" method="post">
    您的姓名:<input name="name"size=10 type=text><br>
    您的个人主页地址:<input name="URL"size=50 type=text value="http://">
        <textarea name="JianYi" clos="20" rows="5">您的建议</textarea><br>
        您的密码:<input type=password name="password1"size=8><br>
    确认密码:<input type=password name="password2"size=8><br>
    所在城市:<select><option selected value ="01">北京</option>
                    <option value="02">上海</option></select><p>
    个人爱好:<input type="checkbox" name="c1">看书
             <input type="checkbox" name="c2">打球
                <input type="checkbox" name="c3">音乐<br>
    性别:<input type="radio" name="r1" checked>男
             <input type="radio" name="r1">女<p>
    <input type="submit" name="s1" value="确定">
    <input type="reset" name="s2" value="重写">
    </form>
</body>
</html>
```

程序运行结果如图 2-3 所示。



图 2-3 表单的应用

2.2 使用 XML 表达数据

XML 是 Internet 环境中跨平台的、依赖于内容的技术，是当前处理结构化文档信息的有力工具。XML 主要用于表达数据，提供了一种描述结构数据的格式。

2.2.1 XML 的概念

可扩展标识语言(eXtensible Markup Language，XML)是由万维网联盟 W3C 定义，它与 HTML 一样，都是 SGML 的子集。XML 被设计用来传输和存储数据。HTML 用来显示数据。XML 在 Web 中起到的作用不亚于作为 Web 基石的 HTML，但 XML 不是 HTML 的替代，而是对 HTML 的补充。

1. SGML、HTML 和 XML

SGML、HTML 是 XML 的先驱。

标准通用标记语言(Standard Generalized Markup Language，SGML)是一种定义电子文档结构和描述其内容的国际标准语言，是所有文档标记语言的起源。SGML 规定了在文档中嵌入描述标记的标准格式，指定了描述文档结构的标准方法，HTML 就是使用固定标签集的一种 SGML 文档。SGML 早于 Web 的诞生，虽然它定义文档的功能强大，但由于标准过严、过于复杂，使得 SGML 直接应用于 Web 的难度非常大，不适于 Web 数据描述。

HTML 是在 SGML 基础上开发出来的应用于 Web 的语言。由于它简单易用，使用成本低，很快成为 Web 的通用语言。随着 Web 的应用越来越深入，人们渐渐觉得 HTML 不够用了，过于简单的语法阻碍了它表达复杂的形式，有限的标记也严重制约着 Web 上的数据交换。HTML 不能表现深层的信息结构，不适于大量文档的存储。HTML 需要下载整个文件，才能开始对文件做搜寻的动作。HTML 的扩充性、弹性、易读性均不佳。

为了解决以上问题，专家们使用 SGML 精简制作，并依照 HTML 的发展经验，开发出一套规则严谨、使用简单的描述数据语言：XML。

XML 结合了 SGML 和 HTML 的优点并消除其缺点。XML 是 SGML 的子集，继承了 SGML 的多数功能，并进行了机能的扩张。XML 掌握了 SGML 的延展性、文件自我描述特性以及强大的文件结构化功能，XML 摒除了 SGML 过于庞大复杂以及不易普及化的缺点。可以说，XML 以 SGML20％的难度实现了 SGML80％的机能，成为 Web 应用中数据传输和交互的重要工具。

2. XML 的树结构

XML 是一种元标记语言。所谓"元标记"，就是 XML 不像 HTML 那样只能使用规定的标记，用户可以自己定义需要的标记。任何满足 XML 命名规则的名称都可以作为标记，这就为不同的应用程序打开了大门。

XML 文档是纯文本，可用文本编辑器创建。XML 文档的文件后缀是. xml。

下面看一个简单的 XML 文档。

例 2-4 一个 XML 文档（bookstore. xml）。

```
<?xml version = "1.0" encoding = " ISO - 8859 - 1"?>
< bookstore >
< book category = "COOKING">
  < title lang = "en"> Everyday Italian </title >
  < author > Giada De Laurentiis </author >
  < year > 2005 </year >
  < price > 30.00 </price >
</book >
< book category = "CHILDREN">
  < title lang = "en"> Harry Potter </title >
  < author > J K. Rowling </author >
  < year > 2005 </year >
  < price > 29.99 </price >
</book >
< book category = "WEB">
  < title lang = "en"> Learning XML </title >
  < author > Erik T. Ray </author >
  < year > 2003 </year >
  < price > 39.95 </price >
</book >
</bookstore >
```

每个 XML 文档都由第一行的 XML 声明开始，定义了 XML 的版本和所使用的编码。第二行是文档的根元素<bookstore>，根元素的子元素是<book>元素。<book>元素有 4 个子元素：<title>、< author>、<year>和<price>，同时，<book>元素还有属性 category。

整个 XML 文档组成一个树型结构。也就是说，XML 文档必须包含根元素，该元素是所有其他元素的父元素。XML 文档中的所有元素形成一棵文档树。这棵树从根部开始，一直扩展到树的最底端。所有元素均可拥有子元素，也都可拥有文本内容和属性。

图 2-4 展示了一棵 XML 文档树。它包括 4 类结点：根结点、元素结点、属性结点和文本结点。根结点表示的是根元素，元素结点用于表示根元素的所有子元素，属性结点用于表示元素的属性，文本结点表示元素的文本内容。

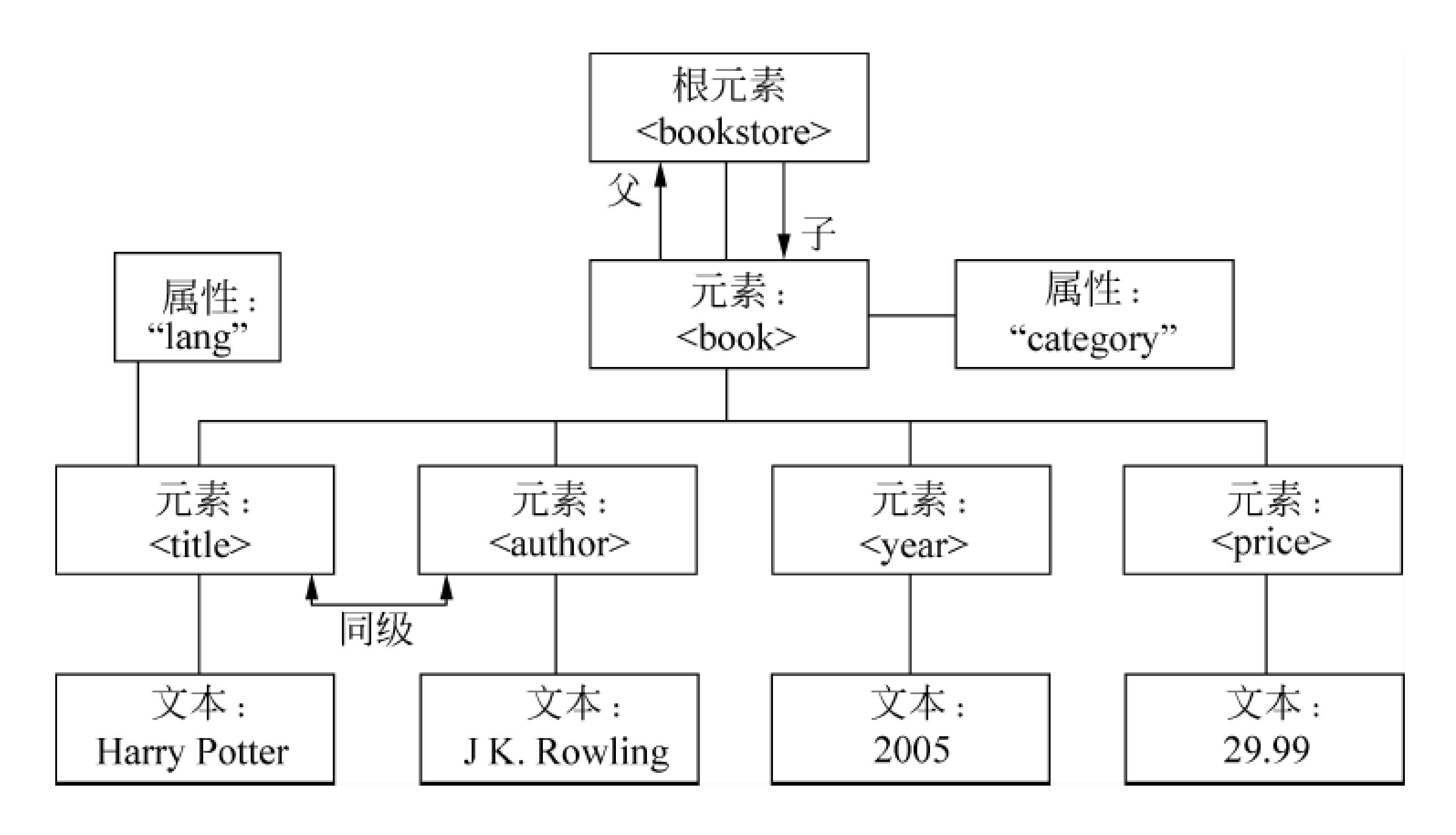


图 2-4　XML 文档树

3. XML 与 HTML 的不同

XML 和 HTML 都来自于 SGML，有着相似的语法，但却有着很大的不同。传统的 HTML 无法表达数据的含义，而这恰恰是电子商务、智能搜索所必需的。HTML 不能描述化学符号、数学公式、音乐符号、矢量图形、影音文件等其他形态的内容。HTML 的可扩展性差，不像 XML 那样可以提供更多的数据操作。

XML 与 HTML 相比，有以下不同：

1）XML 实现内容与形式的分离

HTML 中的数据和表现形式是混合在一起的，当改变内容数据的表现形式时，需更新整个文档。而 XML 文档只包含数据，数据的表现形式由另一个描述格式的文档来定义，因此改变时只要更新该格式文档即可。

2）XML 扩展性比 HTML 强

XML 允许使用者创建个性化的标签，每一个行业或专业领域可以制定特定范围内的标签集，以满足特殊的需要。XML 的扩展性和灵活性允许它描述不同种类应用软件中的数据，从描述 Web 页到数据记录。XML 格式的数据可以用应用软件进行解析。使用者可以用不同的方法处理 XML 数据，而不仅仅是显示它。HTML 只能局限于按一定的格式在终端显示出来。

3）XML 的语法比 HTML 严格

由于 XML 的扩展性强，它需要稳定的基础规则支持扩展，如起始标签和结束标签必须匹配、标签不能交叉嵌套、区分大小写等。HTML 并没有规定标签的绝对位置，也不区分大小写，这些全部由浏览器识别和更正。

4）XML 具有良好的自描述性

HTML 使用固有的标签描述和显示网页内容。XML 不能描述网页的外观和内容，只是描述内容的数据形式和结构。XML 标签拥有特定的语义，更容易被人理解。同时由于 XML 数据是自我描述的，数据不需要有内部描述就能被交换和处理。

2.2.2 XML 的语法规则

1. XML 文档的格式

前面说过，XML 比 HTML 在语法上有着更严格的要求。一个 XML 文档可以包括两个方面：

1）结构良好的（Well-formed）XML 文档

如果某个文档符合 XML 语法规范，那么就说这个文档是“结构良好”的文档。

2）有效的（Validate）XML 文档

有效的 XML 文档是指通过了 DTD 验证的，具有良好结构的 XML 文档。

一个具有良好结构的 XML 文档不一定就是有效的 XML 文档；反之，一个有效的 XML 文档必定是一个结构良好的 XML 文档。

实现一个结构良好的 XML 文档必须满足以下条件：

- XML 文档由“XML 声明”开始；
- 有且仅有一个根元素；
- 所有的 XML 标记必须成对出现，将数据包围在中间；
- 所有的 XML 标记必须对称地嵌套；
- XML 标记对大小写敏感。

所谓“结构良好”是针对 HTML 混乱的语法而言，XML 文档只有格式良好才能被正确地分析和处理。下列是一个合理的 HTML 文档，但却完全不符合 XML 的格式要求。

```
<HTML>
<BODY>
<h2>西安欢迎你<br>
<font color="red" size=4>西安欢迎你</FONT>
<hr>
<b size=6><i>西安欢迎你</b></i>
</body>
</html>
```

还有一点要说明的是，XML 中的空格不会被删节，而是被保留。HTML 则会把多个连续的空格字符合并为一个。在 XML 文档中任何的差错，都会得到同一个结果：网页不能被显示。

有效的 XML 文档是指该 XML 文档符合语义方面的规范。一般来说，如果 XML 文档的语法符合 DTD 或 XML Schema 的定义和规定，那么就是“有效的”XML 文档。

在 DTD 文档中定义了有效的 XML 元素名以及相应的一些规则。当一个 XML 文档与 DTD 文档关联起来以后，此 XML 文档必须遵循 DTD 中定义的规则，否则会视为无效。

XML Schema 的作用与 DTD 类似，但它自身也是一个 XML 文档，从而拥有 XML 的各种特性，因此 XML Schema 要比 DTD 灵活得多。

2. XML 标记的命名规则

XML 标记的命名规则如下：

- 名字中可以包含字母、数字以及其他字母；

- 名字不能以数字或下划线"_"开头；
- 名字不能以字母 xml（或 XML 或 Xml）开头；
- 名字中不能包含空格。

3. XML 声明

XML 声明的作用是告诉浏览器将要处理的文档是 XML 文件。

例如：

```
<?xml version = "1.0"   encoding = "GB2312" ?>
```

其中，version 是必须声明的属性，指明文档遵循哪个版本的 XML 规范。encoding 属性是可选项，指示文档中字符使用的编码标准。常见的有 GB2312、BIG5、UTF-8 等。

2.2.3 验证 XML 的有效性

假设有一个结构良好的 XML 文档 person.xml，现在需要验证它的有效性。

例 2-5 一个结构良好的 XML 文档(person.xml)。

```
<?xml version = "1.0"?>
<person>
<name>George</name>
<sex>Man</sex>
<age>28</age>
</person>
```

如前所述，验证 XML 文档的有效性有 DTD 和 XML Sohema 两种方法，下面分别进行介绍。

1. DTD

文档类型定义(Document Type Definition，DTD)是对 XML 文档所做的规范和约定，指定了一系列 XML 文档必须遵守的规则。实际上，DTD 可以看做是 XML 文档的一个模板。在一个 DTD 中，包含了对 XML 文档所使用的元素、元素间的关系、元素可包含的属性、可使用的实体或符号等的定义规则。

DTD 由许多约定和声明语句构成，这些语句可以包含在 XML 文档内部，称为内部 DTD，也可以独立保存为一个文件，称为外部 DTD。

1）内部 DTD

当 DTD 被包含在 一个 XML 文档中，那么它就包装在一个 DOCTYPE 声明中：

```
<!DOCTYPE 根元素 [元素声明]>
```

下面就是一个带有 DTD 的 XML 文档：

```
<?xml version = "1.0"?>
<!DOCTYPE person [
  <!ELEMENT person (name, sex, age)>
  <!ELEMENT name   (#PCDATA)>
  <!ELEMENT sex    (#PCDATA)>
  <!ELEMENT age    (#PCDATA)>
```

```
]>
<person>
  <name>George</name>
  <sex>Man</sex>
  <age>28</age>
</person>
```

上述 XML 文档中，

! DOCTYPE person 定义此文档是 person 类型的文档。

! ELEMENT person 定义 person 元素有 3 个子元素：name、sex、age。

! ELEMENT name 定义 name 元素为 #PCDATA 类型。

! ELEMENT sex 定义 sex 元素为 #PCDATA 类型。

! ELEMENT age 定义 age 元素为 #PCDATA 类型。

在一个 DTD 中，元素通过元素声明来进行声明。

```
<!ELEMENT 元素名称 (元素内容)>
```

或

```
<!ELEMENT 元素名称 类别>
```

例如：

```
<!ELEMENT person (name, sex, age)>
```

只有 PCDATA 的元素通过括号中的 #PCDATA 进行声明：

```
<!ELEMENT 元素名称 (#PCDATA) >
```

例如：

```
<!ELEMENT name  (#PCDATA)>
```

2）外部 DTD

假如 DTD 位于 XML 文档的外部，那么它应被封装在一个 DOCTYPE 定义中：

```
<!DOCTYPE 根元素 SYSTEM "文件名">
```

上述定义中的“文件名”就是一个外部的 DTD 文档，其后缀为.dtd。

假设有一个 XML 文档，它引用了一个外部 DTD 文档 person.dtd。代码如下：

```
<?xml version="1.0"?>
<!DOCTYPE person SYSTEM "person.dtd">
<person>
<name>George</name>
<sex>Man</sex>
<age>28</age>
</person>
```

下面是 person.dtd 文件的内容。第 1 行定义 person 元素有三个子元素：name、sex、age。第 2～第 4 行定义了 name、sex、age 元素的类型是 ＃PCDATA。

```
<!ELEMENT person (name, sex, age)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT sex (#PCDATA)>
<!ELEMENT age (#PCDATA)>
```

通过 DTD,每个 XML 文档均可包含一个有关自身格式的描述。不同的应用程序只需要定义标准的 DTD,各程序就能够依照 DTD 建立、验证、交换并处理 XML 文档了。

2. XML Schema

XML Schema 是 W3C 开发的一种新的约束 XML 文档的模式,是一种特殊的 XML 文件,遵循 XML 的语法规则。XML Schema 语言也称作 XML Schema 定义(XML Schema Definition,XSD)。

与 DTD 类似,XML Schema 用于描述 XML 文档的合法元素。它弥补了 DTD 的不足之处。例如,DTD 的数据类型有限,当声明一个元素的内容为文本时,声明为＃PCDATA,却不能限制文本的具体类型(如整型、浮点型等)。XML Schema 则可以定义具体的数据类型。XML Schema 不但提供了丰富的数据类型,还允许用户自定义类型。

与 DTD 相比,XML Schema 具有如下优点:

(1) 可以更容易地描述文档结构。

(2) 可以方便地定义数据类型。

(3) 可重用性。

虽然 XML Schema 比 DTD 对数据的限制好,但实现相同功能的 Schema 代码比 DTD 要长很多,DTD 也可以实现 XML Schema 不能实现的功能。它们各有优势。

W3C 规定,一个 XML Schema 文档的根标记必须是 schema,名称空间必须是 http://www.w3.org/2001/XMLSchema,下面是它的基本形式:

```
<?xml version = "1.0" ?>
<xs:schema xmlns:xs = http://www.w3.org/2001/XMLSchema >
            Schema 内容
</xs:schema >
```

上面代码中,第一行以 XML 声明开始,说明这是一个 XML 文件。所有内容都添加在根标记<schema>中。xs 是名称空间的前缀,可以任意定义。

在"schema 内容"中,Schema 标记及属性定义的格式如下:

```
<xs:element name = "标记名称" type = "数据类型"> </xs:element >
```

下面是一个名为 person.xsd 的 XML Schema 文件。

```
<?xml version = "1.0" ?>
<xs:schema xmlns:xs = http://www.w3.org/2001/XMLSchema >
<xs:element name = "person">
    <xs:complexType >
```

```
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="sex" type="xs:string"/>
            <xs:element name="age" type="xs:string"/>
        </xs:sequence>
        </xs:complexType>
    </xs:element>
    </xs:schema>
```

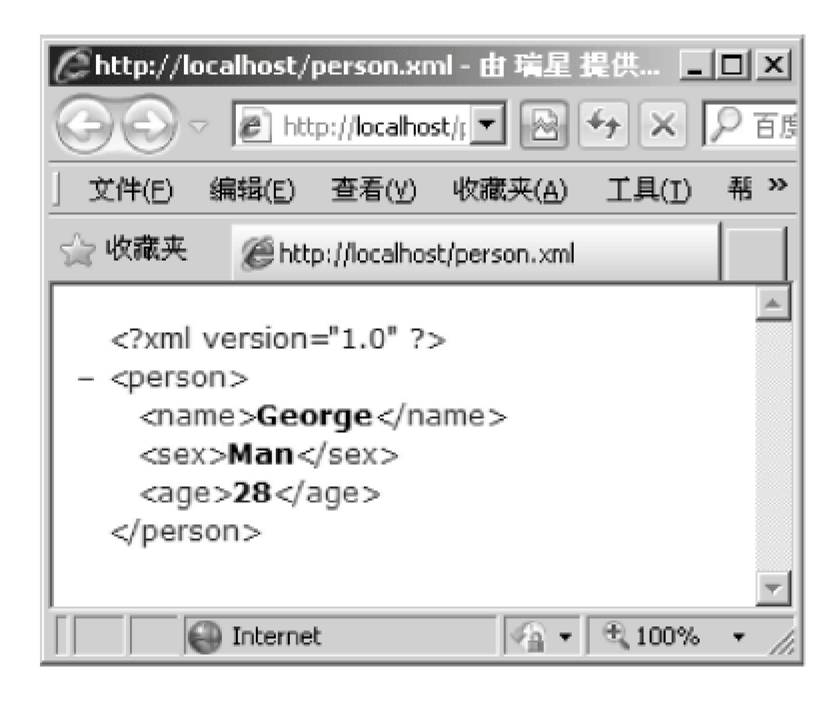
person 元素是一个复合类型，包含其他的子元素。其他元素（name，sex，age）是简单类型，因为它们没有包含其他元素。

当 XML 文档 person.xml 引用这个 XML Schema 文件 person.xsd 时，需要在根标记内声明，代码如下：

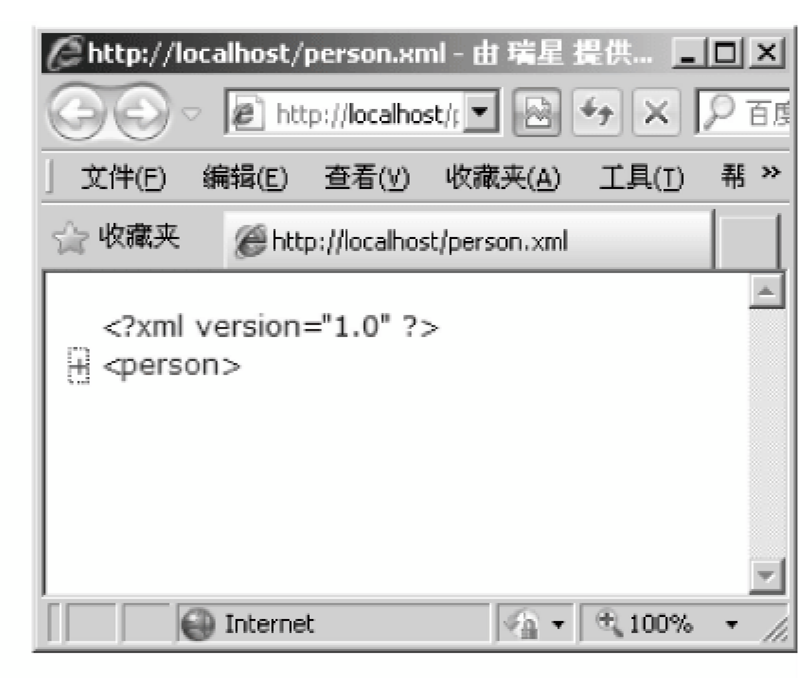
```
<?xml version="1.0"?>
<person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="person.xsd">
    <name>George</name>
    <sex>Man</sex>
    <age>28</age>
</person>
```

2.2.4 XML 文档的显示

原始的 XML 文档可以用 IE 浏览器查看，浏览器将之显示为一棵可折叠的树。以 person.xml 为例，用 IE 浏览器打开的结果如图 2-5 所示。



(a) 折叠前的XML文档



(b) 折叠后的XML文档

图 2-5　在浏览器中打开原始的 XML 文档

XML 文档本身并不包含数据显示的信息，如果希望 XML 文档在浏览器中按照一定的格式显示出来，必须要有一个专门的文件定制 XML 文档的显示样式。这个专门的样式文档可以是 CSS(层叠样式表)或 XSLT(可扩展样式表)。所谓样式表，就是包含一个或多个格式化规则的文档，包含指示 Web 浏览器如何将原文档的结构翻译为一个能够显示的结构的代码。

1. 使用 CSS 在 Web 上显示 XML 数据

层叠样式表(Cascading Style Sheets,CSS)包含一个或多个格式化规则和定义。它控制 XML 文档和 HTML 文档中的标签在浏览器中如何显示。

假设创建一个 CSS 文档 person.css,其内容如下：

```
person
{
  font-size: 24px;
  font-weight: bold;
  display: block;
  color: blue;
}
sex
{
  font-size: 20px;
  font-style: italic;
  text-decoration: underline
}
```

上述文档定义了 XML 文档 person.xml 中根元素 person、子元素 sex 的显示特性。

要按照 person.css 中的定义显示 XML 文档 person.xml 的内容,需要在 XML 文档中添加一条 xml-stylesheet 命令,则此 XML 文档就与 CSS 文件关联起来了。

下面是修改后的 person.xml 文档(person-css.xml)：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="person.css" ?>
<person>
     <to>George</to>
     <sex>Male</sex>
     <age>28</age>
</person>
```

在浏览器中打开该文档,显示的结果如图 2-6 所示。

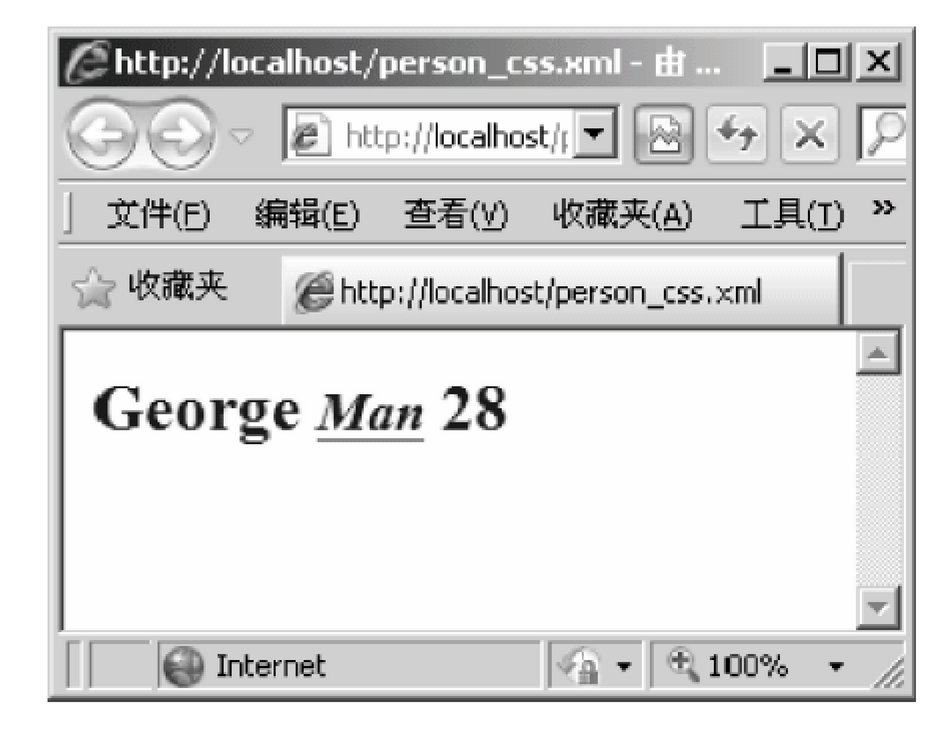


图 2-6 用 CSS 显示 XML 数据

2. 使用 XSLT 显示 XML 数据

可扩展样式表语言(eXtensible Stylesheet Language,XSL)是一种可以将 XML 文档转化成浏览器能够识别的 HTML 文档的语言。通常,XSL 是通过将每一个 XML 元素"翻译"为 HTML 元素,来实现这种转换的。XSL 样式表自身也是一个 XML 文档。

XSL 包括三部分：

- XSLT：一种用于转换 XML 文档的语言。
- XPath：一种用于在 XML 文档中导航的语言。

• XSL-FO：一种用于格式化 XML 文档的语言。

XSLT(eXtensible Stylesheet Language Transformation)是扩展样式表转换语言。XSLT 可将 XML 文档转换为 XHTML 或另一种文本文件。XSLT 通过把每个 XML 元素转换为 HTML 元素来完成这项工作。

假设有一个引用了 XSLT 文件的 XML 文档 person_xslt.xml，其内容如下：

```
<?xml version = "1.0"?>
<?xml - stylesheet type = "text/xsl" href = "person.xslt"?>
<persons>
  <person>
    <name>George</name>
    <sex>Male</sex>
    <age>28</age>
  </person>
  <person>
    <name>Mary</name>
    <sex>Female</sex>
    <age>20</age>
  </person>
</persons>
```

上述 XML 文档中，引用的 XSLT 文档 person.xslt 内容如下：

```
<?xml version = "1.0" encoding = "GB2312"?>
<xsl:stylesheet version = "1.0" xmlns:xsl = "http://www.w3.org/TR/WD - xsl">
<xsl:template match = "/">
<HTML>
<BODY>
<TABLE BORDER = "1">
<TR><TH>姓 名</TH><TH>性 别</TH><TH>年 龄</TH></TR>
<xsl:for - each select = "persons/person">
<TR>
        <TD><xsl:value - of select = "name"/></TD>
        <TD><xsl:value - of select = "sex"/></TD>
        <TD><xsl:value - of select = "age"/></TD>
</TR>
</xsl:for - each>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

可以看到，XSLT 文件以一个 XML 声明开始，使用 xsl:stylesheet 指令声明这是一个样式表文件。<xsl:template match="/">语句表示 XML 的源文档在当前目录下。通过 xsl:for-each 指令查找 XML 文档中路径为 persons/person 的元素，然后使用 xsl:value-of 指令取出此元素的值。通过上述转换，原来的 XML 文档实际上转换成了 HTML 文档。

在浏览器中打开 person_xslt.xml 文件，显示结果如图 2-7 所示。

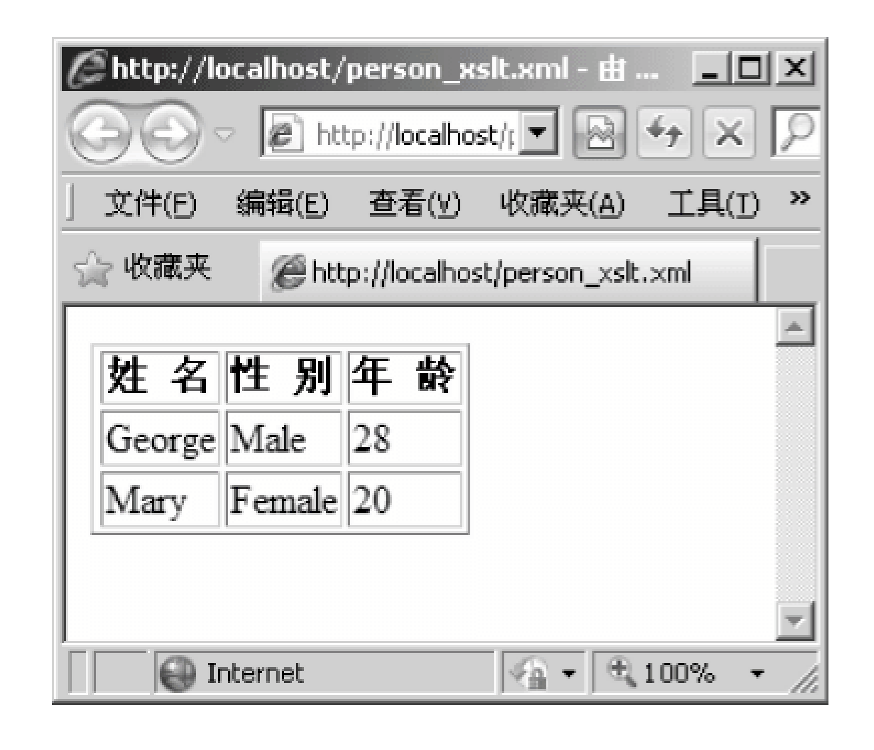


图 2-7　用 XSLT 显示 XML 数据

3. XSLT 和 CSS 的区别

CSS 和 XSLT 虽然都可以格式化 XML 文档，但它们却有很大的不同，如表 2-7 所示。

表 2-7　CSS 和 XSLT 的区别

CSS	XSLT
使用简单	使用复杂
不能排序、添加或删除文档中的元素	可以排序、添加或删除文档元素
消耗内存资源少	使用较多内存和处理器能力
与 XML 语法不同	与 XML 语法相同

CSS 只适合输出比较固定的文档。CSS 的优点是简洁，消耗系统资源少；XSLT 虽然功能强大，但因为要重新索引 XML 结构树，所以消耗内存比较多。因此，常常将它们结合起来使用，如在服务器端用 XSLT 处理文档，在客户端用 CSS 控制显示，这样可以减少响应时间。

2.3　XHTML 和 DHTML

2.3.1　XHTML

```
XHTML = XML + HTML
```

XHTML(eXtensible HyperText Markup Language)是遵循 XML 规范的 HTML，比 HTML 要严格，是一种增强了的 HTML。2000 年，W3C 公布了 XHTML 1.0 版本。XHTML 1.0 是在 HTML 4.01 的基础上优化和改进的新语言，其目的是基于 XML 应用，它的可扩展性和灵活性更能适应未来网络应用的需求。

1. XHTML 代码的基本规范

(1) 所有的标签必须严格配对，即使是空元素也得封闭。

在 HTML 中，用户打开的许多标签可以不配对，但在 XHTML 中却是非法的。XHTML 要求有严谨的结构，所有标签必须关闭。对于单独不成对的标签，在标签最后加一个“/”来关闭它。

例如：

```
<img src="../logo.gif" height="80" width="200" />
<br />
```

（2）所有的标签名和属性名必须小写。

XHTML 对大小写是敏感的，<title>和<TITLE>是不同的标签。XHTML 要求所有的标签和属性名都必须使用小写字母。

（3）所有的属性值必须用双引号括起来。

在 HTML 中，用户可以不给属性值加引号，但在 XHTML 中必须加引号。例如，在 HTML 中可以有这样的代码：

```
<input name=apple type=checkbox value=apple>
```

在 XHTML 中必须写为：

```
<input name="apple" type="checkbox" value="apple" />
```

（4）所有的标签必须严格嵌套。

在 HTML 中可以这样编写代码：

```
<p><b>hello</p></b>
```

按照 XHTML 的要求必须改为：

```
<p><b>hello</b></p>
```

（5）每个属性必须赋值。

XHTML 规定所有属性都必须有一个值，没有值的就重复自身。例如，在 HTML 中可以这样编写代码：

```
<td nowarp><input type="checkbox" name="apple" value="big" checked>
```

按照 XHTML 的要求必须改为：

```
<td nowarp="nowarp"><input type="checkbox" name="apple" value="big"
checked="checked" /></td>
```

2. XHTML 的基本结构

XHTML 文档是一种纯文本格式的文件，XHTML 文档的基本结构为：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>网页标题</title>
</head>
```

```
<body>网页内容</body>
</html>
```

一个XHTML文档由三部分组成：声明、头部、主体。

文档的最前面是一个DOCTYPE声明，定义了文档使用的DTD版本、类型、下载地址等。本例中定义了文档使用的语言版本是XHTML 1.0。文档类型是Transitional。DTD下载地址是http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd。

<html>标签必须带xmlns属性，表明使用的命名空间。

文档头部是<head>…</head>之间的部分，主要用来定义文档的相关信息，如文档标题、说明信息、样式定义、字符集等。

文档主体是<body>…</body>之间的部分，其内容就是要展示给用户的部分。

2.3.2 DHTML

```
DHTML = HTML/XHTML + CSS + JavaScript
```

动态的HTML(Dynamic HTML，DHTML)不是一种网页设计语言、W3C技术标准或规范。DHTML是一种使HTML页面具有动态特性的艺术。DHTML是一种创建动态交互站点的技术集。

对大多数人来说，DHTML意味着HTML、样式表和JavaScript的组合。

- HTML/XHTML用于定义文档结构；
- CSS确定信息的外在表现形式；
- JavaScript则用于编程以便动态操控CSS和HTML。

第3章将介绍CSS，第4章将介绍JavaScript。

2.4 习题与上机练习

1. 填空题

(1) HTML网页文件的标记是________，网页文件的主体标记是________，页面标题的标记是________。

(2) 表格的标签是________，单元格的标签是________。表格的宽度可以用百分比和________两种单位来设置。

(3) 表单对象的名称由________属性设定；提交方法由________属性指定；若要提交大量数据，则应采用________方法；表单提交后的数据处理程序由________属性指定。

(4) DTD(文档类型定义)是对________文档所做的规范和约定。

2. 选择题

(1) 下面(　　)属性不是文本的标签属性。

A. nbsp　　B. align　　C. color　　D. face

(2) 下列(　　)表示的不是按钮。

A. type="submit"　　B. type="reset"

C. type="image"　　D. type="button"

(3) 在 DHTML 中把整个文档的各个元素作为对象处理的技术是(　　)。

A. HTML　　B. CSS

C. DOM　　D. Script(脚本语言)

(4) 下列 HTML 标记中,属于非成对标记的是(　　)。

A. 　　B. 　　C. <P>　　D. <font>

(5) 若设计网页的背景图形为 bg.jpg,以下标记中正确的是(　　)。

A. <body background="bg.jpg">　　B. <body bground="bg.jpg">

C. <body image="bg.jpg">　　D. <body bgcolor="bg.jpg">

(6) 若要在页面中创建一个图形超链接,要显示的图形为 myhome.jpg,所链接的地址为 http://www.pcnetedu.com,以下用法中正确的是(　　)。

A. myhome.jpg

B. <img src="myhome.jpg">

C. <img src="myhome.jpg">

D. <a href =http://www.pcneredu.com><img src="myhome.jpg">

(7) 要将页面当前位置定义成一个名为 vbpos 的标签,定义方法正确的是(　　)。

A. 　　B. vbpos

C. <a name=vbpos>　　D.

(8) 如果需要在 XML 文件中显示简体中文,那么 encoding=(　　)。

A. GB2312　　B. BIG5　　C. UTF-8　　D. UTF-16

(9) 在进行属性声明时,错误的是(　　)。

A. 标记中可以使用多个属性

B. 属性可以出现在起始标记或结束标记中

C. 属性名是大小写敏感的

D. 属性值必须用引号引上

(10) 不符合格式良好的 XML 文档要求的是(　　)。

A. 文档可以有一个以上的根元素　　B. 有符合规范的声明语句

C. 每个元素有正确的起始、结束标记　　D. 元素正确嵌套,没有交叉现象出现

3. 简答题

(1) 比较 XML 与 HTML 的异同。

(2) 简述 XHTML 和 DHTML 的含义。

(3) 比较 XSLT 和 CSS 的区别。

4. 上机练习

(1) 制作一个带有网页内部书签链接的文件,包括书签名、作者简介、成果、爱好兴趣。

(2) 用 XML 完成对一本书的描述(要求有书名、作者、单价、出版社、出版日期等,其中 ISBN 是属性),定义命名空间为 http://www.w3.org/2001/XML schema。

第 3 章 利用 CSS 布局网页

CSS 可以定义 HTML 文档和 XML 文档在浏览器中显示的样式。第 2 章已经讲过如何利用 CSS 文件在浏览器中显示 XML 数据。其实,CSS 更多地用于定义 HTML 网页在浏览器中的显示效果。掌握 CSS 技术的基本用法和编程技巧,对学习网页设计至关重要。

3.1 CSS 概 述

CSS 是一种标记性语言,用于控制网页样式,并允许将网页内容与显示样式分离,为网页里的元素创建在浏览器中的表现样式。CSS 以 HTML 语言为基础,提供了丰富的格式化功能,如字体、颜色、背景和整体排版等。

例 3-1 一个简单的 HTML 网页(welcome. htm)。

```
<!DOCTYPE html PUBLIC " - //W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1 - transitional.dtd">
< html xmlns = "http://www.w3.org/1999/xhtml">
< head>
< meta http - equiv = "Content - Type" content = "text/html; charset = gb2312" /> </head>
< body>
< h1 >欢迎你到西安来!</h1 >
< h3 >欢迎你常到西安来!!</h3 >
</body >
</html >
```

当在浏览器中打开这个网页时,浏览器将以默认的样式显示网页内容,如图 3-1 所示。
现在修改例 3-1 的 HTML 文件,加入 CSS 代码后,如下所示(welcome_css. htm):

```
<!DOCTYPE html PUBLIC " - //W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1 - transitional.dtd">
< html xmlns = "http://www.w3.org/1999/xhtml">
< head>
< meta http - equiv = "Content - Type" content = "text/html; charset = gb2312" />
< style type = "text/css">
h1 {font - style:italic;color:red;display:inline; }
</style >
</head >
< body>
< h1 >欢迎你到西安来!</h1 >
```

```
<h3 style = "color:black;font - weight:200">欢迎你常到西安来!!</h3>
</body>
</html>
```

在浏览器中打开 welcome_css. htm，显示结果如图 3-2 所示。可以看出，原来代码中的 <h1>、<h3>标记在加了 CSS 样式说明以后，显示的效果就不同了。

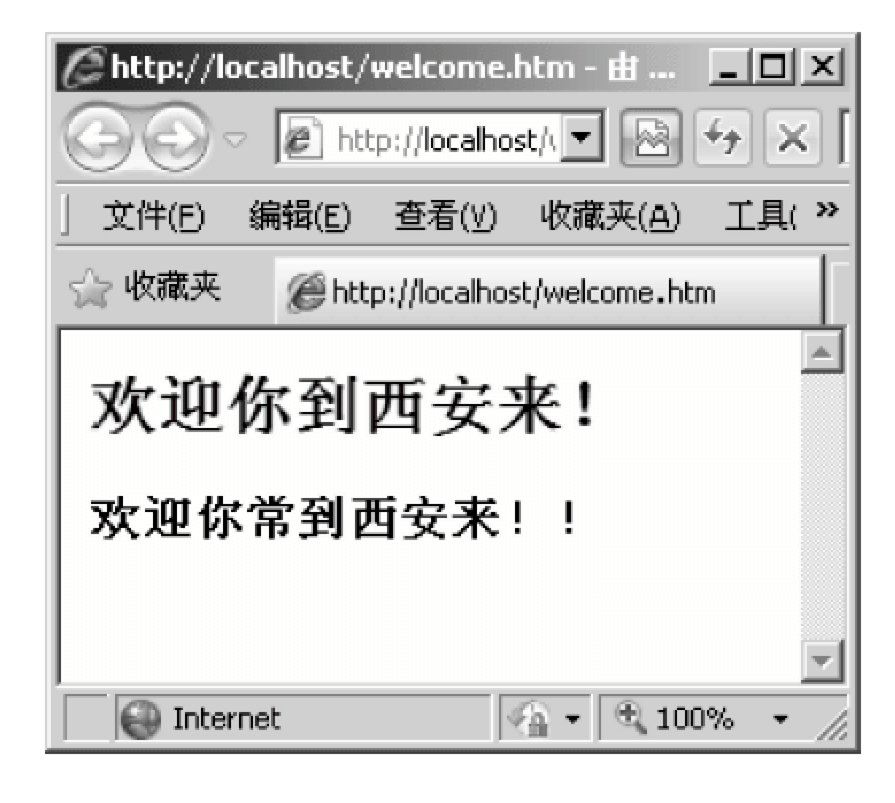


图 3-1 浏览器以默认样式显示 HTML 网页

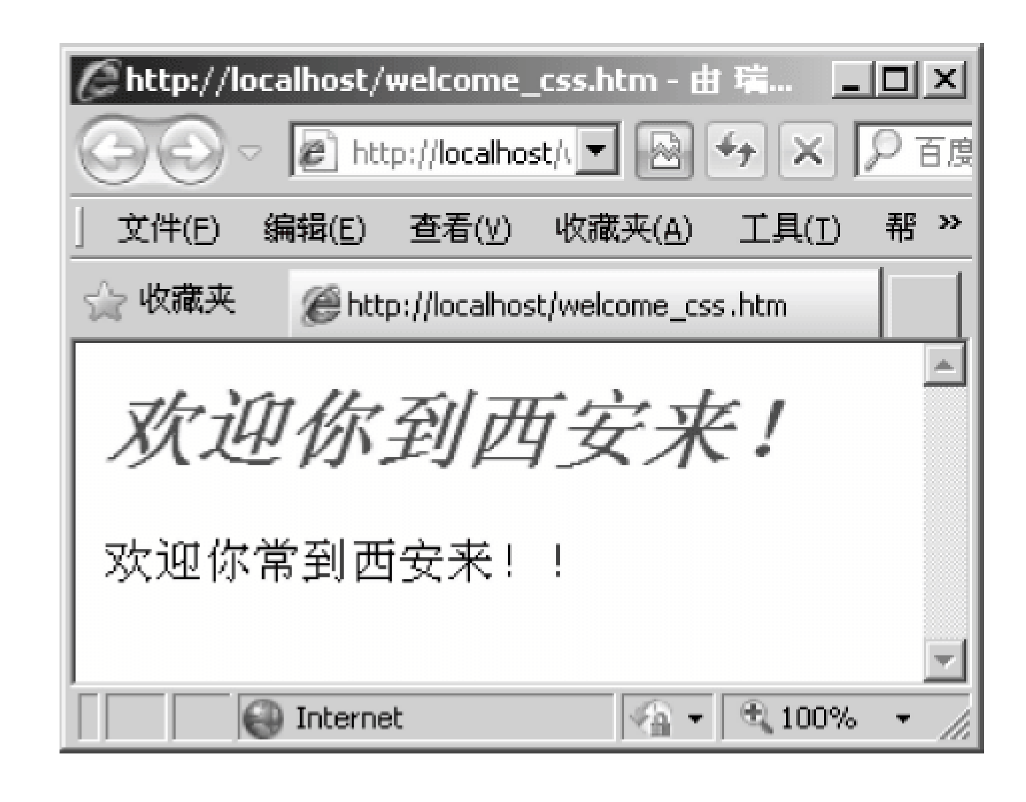


图 3-2 浏览器按指定样式显示 HTML 网页

CSS 的主要特点概括如下：

(1) CSS 与 HTML 的关系体现了“内容结构和格式控制相分离”的原则。

(2) CSS 更容易控制页面的布局。使用外部链接的 CSS 文件，可以方便地修改网页的显示效果而不必修改网页本身；也可以同时定义多个网页的显示风格。

(3) 可以制作更小、下载更快的网页。CSS 简化了 HTML 网页的格式代码，外部的 CSS 样式表还会保存在缓存里，加快了下载显示的速度。

(4) 可以更快、更容易地维护及更新大量的网页。当需要修改一个样式规则时，所有应用了此规则的网页就会自动更新显示效果，大大减少了重复劳动的工作量。

(5) 良好的浏览器兼容性。所有的主流浏览器都支持 CSS。

3.2 在 HTML 中使用 CSS

在 HTML 文档中使用 CSS 样式表，主要分为三种方式：

- 内嵌样式(Inline Style)；
- 内部样式表(Internal Style Sheet)；
- 外部样式表(External Style Sheet)。

除此之外，还可以使用@import 指令导入外部样式表文件。

下面分别对这几种方式加以说明。

1. 内嵌样式

内嵌样式是指对<body>和</body>之间的 HTML 标签，直接设置标签的 style 属性为 CSS 代码。它只对所在的 HTML 标签有效。其格式为：

```
<标签名 style="CSS 代码"></标签名>
```

例如：

```
<html>
<body>
<h3 style="color:black;font-weight:200">西安欢迎你</h3>
<P style="font-size:20pt; color:red">西安欢迎你</p>
</body>
</html>
```

2. 内部样式表

内部样式表是指在HTML文档的<head>标签内部，定义一对<style></style>标签。它只对当前所在的网页有效。其格式如下：

```
<style type="text/css"> CSS 代码 </style>
```

例如：

```
<head>
<style type="text/css" media="screen,projection">
<!--
    p { font-size:20pt; color:blue; font-family:宋体; list-style-type:circle; }
-->
</style>
</head>
```

3. 外部样式表

当多个HTML网页使用同样的CSS规则时，可以将这些CSS规则放在一个以css为后缀的独立文件中，然后在网页里使用<link>引用这个CSS文件。其格式如下：

```
<link type="text/css" rel="stylesheet" href="CSS 文件的 URL" />
```

属性rel是relation的缩写，表示HTML文件与所链接的对象之间的关系；属性href可以是一个完整的URL，指定CSS文件的位置。

例如：

```
<HTML>
<HEAD><link href="../css_tutorials/home.css" rel="stylesheet" type="text/css"></HEAD>
<BODY>
<H1 class="mylayout"> 这个标题使用了 Style</H1>
<H1> 这个标题没有使用 Style </H1>
</BODY>
</HTML>
```

将HTML页面本身和CSS样式分离为不同的文件，实现了页面框架代码和页面布局CSS代码的完全分离，使得网页的前期制作和后台维护都非常方便。

使用外部样式表，相对于内嵌样式和内部样式表，有以下优点：

- 样式代码可以复用。一个外部 CSS 文件，可以被很多网页共用。
- 便于修改。如果要修改样式，只需要修改 CSS 文件，而不需要修改每个网页。
- 提高网页显示的速度。如果样式写在网页里，会降低网页显示的速度，如果网页引用一个 CSS 文件，这个 CSS 文件多半已在缓存区，网页显示的速度就比较快。

4. 使用@import 指令引入外部 CSS 文件

使用@import 声明可以将一个外部 CSS 文件输入到另外一个 CSS 文件中。格式如下：

```
<style type="text/css">
    @import " CSS 文件的 URL";
</style>
```

例如：

```
<style type="text/css">
    <!—
    @import url (http://www.ccut.edu.cn/style.css)
    @import url (/css/style.css)
    body { background:red ;color:green}
    ->
</style>
```

5. 样式表的优先级顺序

样式表允许以多种方式定义样式信息。样式可以定义在单个的 HTML 标签内部、HTML 页的<head>标签内部、或在一个外部 CSS 文件中。甚至可以在同一个 HTML 文档内部引用多个外部样式表。

当同一个 HTML 元素被不止一个样式定义时，会使用哪个样式呢？

一般而言，所有的样式会根据下面的规则层叠于一个新的虚拟样式表中，它们的优先顺序如下：

(1) 浏览器默认(Browser Default)，优先级最低。

(2) 外部样式表(Extenal Style Sheet)。

(3) 内部样式表(Internal style sheet，位于<head>标签内部)。

(4) 内嵌样式(Inline style，在 HTML 元素内部)，优先级最高。

内嵌样式拥有最高的优先权，样式的优先级依次是内嵌、内部、外部、浏览器默认。

假设内嵌样式中有 font-size:30pt，而内部样式中有 font-size:12pt，那么内嵌样式就会覆盖内部样式。

例如：

```
<HTML>
<HEAD>
        <TITLE>Cascading Order</title>
```

```
        <STYLE TYPE = "text/css"> p {font - size:12pt}</STYLE>
</HEAD>
<BODY>
        <p style = "font - size:30pt">西安欢迎你!</p>
</BODY>
</HTML>
```

上述代码中，段落标记<p>的内嵌样式覆盖了内部样式表，因此，显示的字体大小是30pt，而不是12pt。

3.3 CSS 基本语法

3.3.1 样式规则的基本结构

一个样式(Style)的基本结构由三部分组成：选择器(Selector)、属性(Property)、属性值(Value)。其格式为：

```
selector { property: value; property: value; … }
```

selector：当定义一条样式规则时，必须指定这条规则作用的 HTML 元素。

property：指定要被修改的样式风格的名称，即 CSS 属性。

value：属性 property 的值。

例如：

```
p {color:blue}
```

这里，p 就是选择器；color 就是属性；blue 就是属性值。

HTML 中所有的标签都可以作为选择器。如果想在 Style 中定义多个属性，两个属性之间必须用分号加以分隔。

例如：

```
p { text - align:center; color:red }
```

为了提高 Style 代码的可读性，也可以分行写：

```
p
{
    text - align: center;
    color: red
}
```

当多个选择器都有相同的属性和属性值时，可以将多个选择器之间用逗号分隔。下面的例子是将所有正文标题(<h1>到<h6>)的字体颜色都变成红色。

```
h1,h2,h3,h4,h5,h6
{
```

```
    text - align: center;
    color: blue
  }
```

为了方便理解 CSS 代码，还可以写 CSS 代码注释。CSS 代码注释以/* 开头，以 */结束。

例如：

```
  p
  {
    text - align: center;
    /* 居中显示 */
    color: black;
    font - family: arial
  }
```

3.3.2 CSS 选择器

HTML 页面中的标记都是通过不同的 CSS 选择器去控制的。每条 CSS 规则都至少包含一个选择器。常用的 CSS 选择器主要包括以下几种：

- HTML 标签选择器；
- 类选择器；
- id 选择器；
- 伪类选择器；
- 派生选择器。

下面分别介绍这几种选择器，以及它们使用的优先级。

1. HTML 标签选择器

一个 HTML 页面由很多不同的标签组成，标签选择器直接声明哪些标签采用哪种 CSS 样式。格式如下：

```
标签名 { 属性:属性值; 属性:属性值; … }
```

例如：

```
h1 {color:red; font - size:15pt }
h1, h2, h3, h4 {color:red }
```

2. 类选择器

在同一 HTML 文档中，当同一标签需要使用不同的样式，或同一样式被不同标签使用时，需要用到类选择器。例如，当显示论文的摘要和正文时，需要用到不同的字体，但两者可能都使用了<p>标签，如果仅使用简单的标签选择器，则无法区别对待不同部分的段落样式，而类选择器则很好地解决了这个问题。

(1) 定义 Class selector 的格式为：

```
标签名.类名{ 属性:属性值; 属性:属性值; … }
```

或

```
.类名{ 属性:属性值; 属性:属性值; … }
```

(2) 引用 Class selector 的格式为：

```
<标签名 class="类名">
```

下面按两种不同使用情况分别举例(注意,类名是可以任意定义的)。

① 同一标签需要使用不同的样式。

例如,段落<p>有两种样式：一种是居中对齐；另一种是居右对齐。定义样式如下：

```
p.center {text-align:center}
p.right {text-align:right}
```

其中 right 和 center 就是两个 class。引用这两个 class 的示例代码如下：

```
<p class="center">这一段居中显示</p>
<p class="right">这一段是居右显示</p>
```

② 同一样式被不同标签使用。

例如,直接用“.”加上类名作为一个选择器。代码如下：

```
.center {text-align: center}
```

这种通用的 Class Selector 没有 HTML 标签的限制,可以用于不同的标签。

```
<h1 class="center">这个标题居中显示</h1>
<p class="center">这个段落居中显示</p>
```

3. id 选择器

id 选择器可以为标有特定 id 的 HTML 元素指定特定的样式。在同一个 HTML 页面中,id 是唯一的,只能定义一次。如果多次使用同一个 id 名称,会导致与其他要求唯一 id 的应用程序发生冲突。因此,与类选择器不同,在一个 HTML 文档中,id 选择器会使用一次,而且仅一次。

要定义一个 id 选择器,要在 id 名称前加一个“#”号。

(1) 定义 id 的格式为：

```
#ID 名 { 属性:属性值; 属性:属性值; …}
```

(2) 引用 id 的格式为：

```
<标签名  id="ID 名">
```

例如：

```
<style Type = "text/css">
<! --
   #red {color: red}
   #blue {color: blue}
 ->
</style>
<p id = "red">这个段落是红色</p>
<p id = "blue">这个段落是蓝色</p>
```

4. 伪类选择器

有一些特殊的 HTML 元素可以拥有不同的状态。例如,用于定义超链接的<a>标签就可以处于“未被访问”、“已被访问过”、“鼠标悬浮其上”等几种状态。

对于这种元素,CSS 使用伪类选择器来给其不同的状态定义样式。

伪类选择器的格式为:

```
HTML 元素:伪类名 { 属性:属性值; 属性:属性值; … }
```

常用的伪类如下:

- A:link　超链接的正常状态(没有任何动作前);
- A:visited 访问过的超链接状态;
- A:hover 光标移动到超链接上的状态;
- A:active 选中超链接时的状态;
- P:first-line 段落中的第一行文本;
- P:first-letter 段落中的第一个字母。

例如:

```
<style type = "text/css">
a:link {color: #FF0000}            /* 未被访问的链接 红色 */
a:visited {color: #00FF00}         /* 已被访问过的链接 绿色 */
a:hover {color: #FFCC00}           /* 鼠标悬浮在上的链接 橙色 */
a:active {color: #0000FF}          /* 鼠标点中激活链接 蓝色 */
</style>
```

此外,还有一种方式,就是伪类可以与 CSS 类配合使用。其格式为:

```
HTML 元素.类名:伪类名 { 属性:属性值; 属性:属性值; … }
```

例如:

```
<style type = "text/css">
a.c1:link {color: #FF0000}            /* 未被访问的链接 红色 */
a.c1:visited {color: #00FF00}         /* 已被访问过的链接 绿色 */
a.c1:hover {color: #FFCC00}           /* 鼠标悬浮在上的链接 橙色 */
a.c1:active {color: #0000FF}          /* 鼠标点中激活链接 蓝色 */
</style>
```

注意：由于 CSS 优先级的关系（后面比前面的优先级高），在写<a>的 CSS 代码时，一定要按照 a:link、a:visited、a:hover、a:actived 的顺序书写。

5. 派生选择器

派生选择器允许根据文档的上下文关系来确定某个标签的样式。通过合理地使用派生选择器，可以使 HTML 代码变得更加整洁（有的资料也叫上下文选择器、关联选择器、后代选择器、父子选择器等）。

如果想对特定 HTML 元素中的子元素设定样式，这时就可以使用派生选择器。格式如下：

```
父元素 子元素
{ 属性:属性值; 属性:属性值; … }
```

说明：父元素和子元素之间用空格隔开，甚至子元素后面还可以有孙子元素。它们从左往右，依次细化，最后锁定要控制的元素标签。

例如：

```
<html>
<head>
<title>Class Selector</title>
      <style TYPE="text/css">p em {color:red}</style>
</head>
<body>
<p>段落中用 em 强调的字是<em>红色</em>的</p>
<h3>标题中用 em 强调的字<em>不是红色</em>的</h3>
</body>
</html>
```

上述代码中，为嵌入<p>元素中的子元素定义了样式：p em{color: red}。

在这里，p em 就叫做 Contextual Selector，表示在<p>里面定义了一个用 em 标记的样式，即{color: red}。因此，只有在<p>里面用标记的字体才是红色，而<h3>中用标记的字却不是红色。

再看看下面的 CSS 规则：

```
<style TYPE="text/css">
  strong { color: red; }
  h2 { color: red; }
  h2 strong { color: blue; }
</style>
```

下面是它施加影响的 HTML 语句：

```
<body>
  <p>The strongly emphasized word in this paragraph is<strong>red</strong>.</p>
  <h2>This subhead is also red.</h2>
  <h2>The strongly emphasized word in this subhead is<strong>blue</strong>.</h2>
</body>
```

6. CSS 选择器的优先级次序

一般而言,CSS 选择器越特殊,它的优先级越高。选择器指向的越准确,它的优先级就越高。因此,通常选择器的优先级是:

派生选择器 > ID 选择器 > 类选择器 > HTML 标签选择器

3.3.3 样式规则的继承

样式规则的继承是指嵌套的 HTML 子元素会继承外层的父元素所设置的样式规则。例如,有这样的 CSS 规则:

```
<style TYPE = "text/css">
  body { font - family: Verdana, sans - serif; }
  p { font - family: Times, "Times New Roman", serif;   }
</style>
```

根据上面的第一条规则,页面<body>元素内的所有子元素将使用 Verdana 字体(假如系统中存在该字体的话),也就是说,这些子元素将继承父元素<body>所拥有的一切属性(子元素诸如 p, td, ul, ol, li, dl, dt 和 dd 等),子元素的子元素也一样。但是,假如希望段落的字体是 Times,那么就创建一个针对 p 的特殊规则(即第二条规则),这样它就会摆脱父元素的规则。

3.4 常见的样式属性

常见的样式属性包括字体、文本、背景、边框、边距、列表样式、定位属性等。下面对这些属性进行简要介绍。

1. 字体属性

CSS 字体属性定义文本的字体系列、大小、加粗、风格(如斜体)等,如表 3-1 所示。

表 3-1 CSS 字体属性

属　　性	描　　述
font	可设置字体的所有属性
font-family	设置字体系列
font-size	设置字体的尺寸
font-style	设置字体风格,取值为 normal/italic/oblique
font-variant	字体变体,取值为 normal/small-caps
font-weight	设置字体的粗细,默认为 normal

2. 文本属性

CSS 文本属性可定义文本的外观。通过文本属性,可以改变文本的颜色、字符间距、对齐文本、装饰文本、对文本进行缩进等,如表 3-2 所示。

3. 背景属性

CSS 允许应用纯色作为背景,也允许使用背景图像创建复杂的效果。CSS 的背景属性如表 3-3 所示。

表 3-2　CSS 文本属性

属　　性	描　　述	属　　性	描　　述
color	设置文本颜色	text-align	文本对齐
direction	设置文本方向	vertical-align	文本的垂直对齐方式
line-height	设置行高	letter-spacing	设置字符间距
text-indent	文本首行缩进	word-spacing	设置字间距
text-decoration	设定文本划线		

表 3-3　CSS 背景属性

属　　性	描　　述	属　　性	描　　述
background-color	设定背景颜色	background-attachment	图片是否跟随内容滚动
background-image	设定背景图片	background-position	背景图片的最初位置
background-repeat	背景图片是否重复	background	可设置背景的所有相关属性

4. 边框属性

通过使用 CSS 边框属性，可以在文本周围创建出效果出色的边框，并且可以应用于任何元素。元素的边框就是围绕元素内容和内边距的一条或多条线。每个边框有三个方面：宽度、样式以及颜色，如表 3-4 所示。

表 3-4　CSS 边框属性

属　　性	描　　述	属　　性	描　　述
border-style	设定上下左右边框的风格	border-color	设定上下左右边框的颜色
border-width	设定上下左右边框的宽度	border	可设置边框的所有属性

5. 边距属性

边距属性设置页面中一个元素所占空间的边缘到相邻元素之间的距离，如表 3-5 所示。

表 3-5　CSS 边距属性

属　　性	描　　述	属　　性	描　　述
margin-left	设定左边距的宽度	margin-bottom	设定下边距的宽度
margin-right	设定右边距的宽度	margin	可以设置上下左右边距的属性
margin-top	设定上边距的宽度		

例 3-2　设置元素的上下左右边距（宽度相同）。

```
<HTML>
  <HEAD>
       <TITLE>CSS 边距属性 margin</TITLE>
       <STYLE type="text/css">
           .D1{border:1px solid #FF0000;}
           .D2{border:1px solid gray;}
           .D3{margin:1cm;border:1px solid gray;}
       </STYLE>
  </HEAD>
```

```
<BODY>
    <DIV CLASS="D1"><DIV CLASS="D2">没有 margin</DIV></DIV>
    <P>上面的 div 没有设置边距属性,仅设置了边框属性(border).</P>
    <HR>
    <P>下面的 div 设置了边距属性,边距 1cm,表示上下左右的边距都为 1cm.</P>
    <DIV CLASS="D1"><DIV CLASS="D3">margin 设为 1cm</DIV></DIV>
</BODY>
</HTML>
```

在浏览器中打开后的效果如图 3-3 所示。

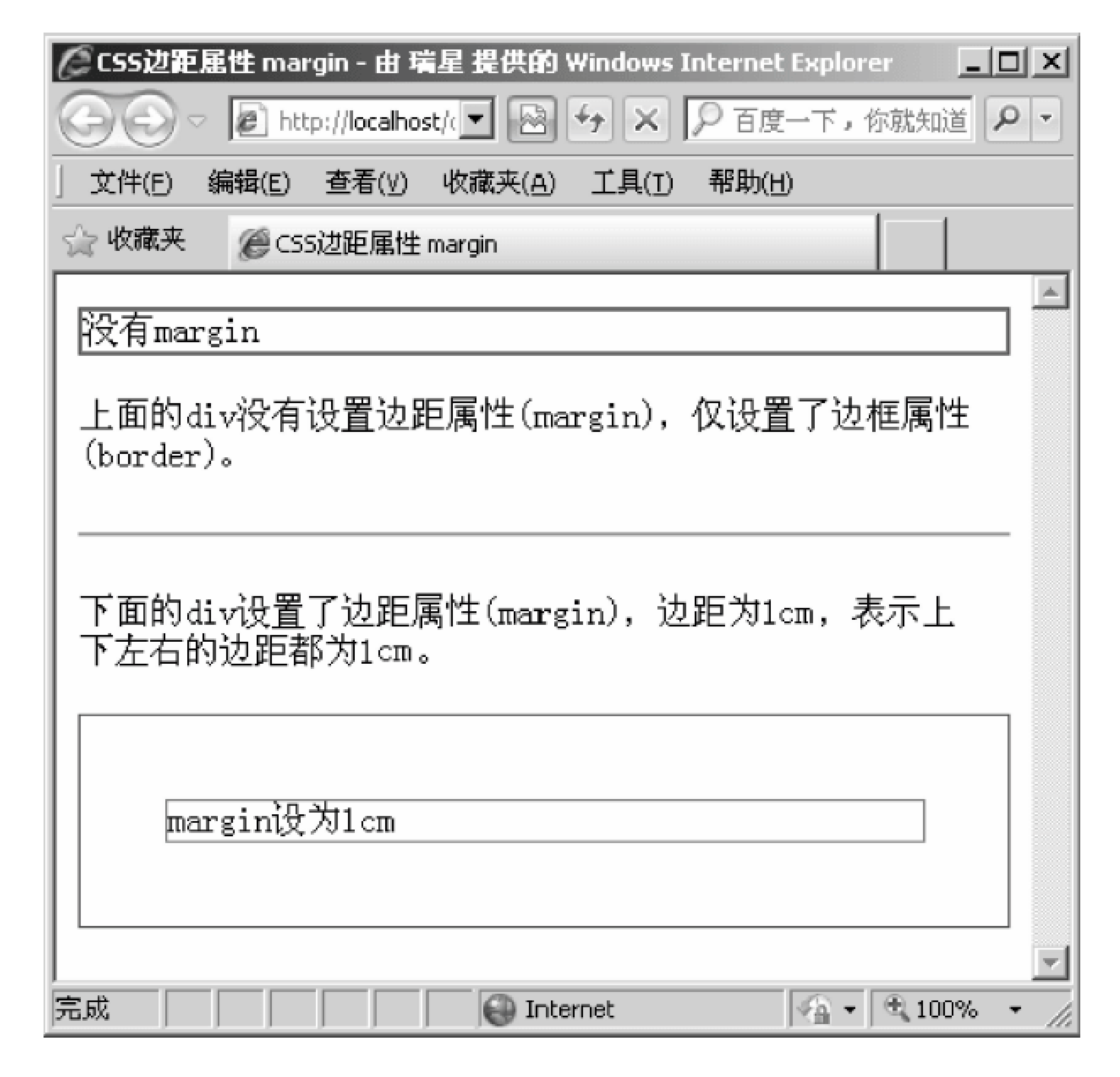


图 3-3 设置元素的上下左右边距

6. 列表样式属性

CSS 列表样式属性可以放置、改变列表项标志,或将图像作为列表项标志,如表 3-6 所示。

表 3-6 CSS 列表样式属性

属性	描述	属性	描述
list-style-type	设置列表样式类型	list-style-image	设置列表样式图片
list-style-position	设置列表样式位置	list-style	设置列表样式的所有属性

例如,设定列表样式相关属性的代码如下:

```
<html>
<head><title>列表样式 list-style</title>
<style type="text/css">
ul {list-style:circle inside url(../images/dot02.gif)}
</style>
</head>
```

```
<body>
<ul><li>茶</li><li>咖啡</li><li>可乐</li></ul>
</body>
</html>
```

7. 间隙属性

间隙属性(Padding)是用来设置元素内容到元素边界的距离，如表 3-7 所示。

表 3-7　CSS 间隙属性

属　性	描　述	属　性	描　述
padding-left	设定左间隙的宽度	margin-bottom	设定下间隙属性
padding-right	设定右间隙的宽度	Padding	同时设定上下左右间隙属性
padding-top	设定上间隙属性		

例如，可以为上下左右间隙设置相同的宽度。代码如下：

```
.d1 {padding:1cm}
```

也可以分别设置间隙，顺序是上、右、下、左。代码如下：

```
.d1 {padding:1cm 2cm 3cm 4cm}
```

表示上间隙为 1cm，右间隙为 2cm，下间隙为 3cm，左间隙为 4cm。

8. 定位属性

CSS 定位 (Positioning) 属性可以对元素进行定位。利用这些属性，可以定义元素的位置、可见性、移动元素、堆叠元素等。定位的基本思想很简单，就是允许定义元素框相对于其正常位置的定位，或相对于父元素、另一个元素甚至浏览器窗口本身的位置，如表 3-8 所示。

表 3-8　CSS 定位属性

属　性	描　述
position	定义元素的定位方式(absolute/fixed/relative/static/inherit)
top	定义元素的顶部边缘(元素顶部的垂直位置)
right	定义元素的右边缘
bottom	定义元素的底部边缘
left	定义元素的左边缘(元素左边的水平位置)
overflow	设置当元素的内容溢出其区域时发生的事情
clip	设置元素的形状，即规定一个元素的可见尺寸
vertical-align	设置元素的垂直对齐方式
z-index	设置元素的堆叠顺序

3.5　CSS 盒子模式

互联网上的大部分网页都采用一种“块”状结构，这些块之间可以相互嵌套，通过“块”的排列与定位实现网页的总体布局。这种块状布局通常不是用传统的表格来排版，而是采用

DIV 来进行编排，这就是 CSS 盒子模式(Box Model)。

传统的表格排版是通过大小不一的表格和表格嵌套来定位排版网页内容，改用 CSS 排版后，就是通过由 CSS 定义的大小不一的盒子和盒子嵌套来编排网页。这种方式排版可以实现网页内容与表现的分离，使得网页代码简洁、更新方便，能兼容更多的浏览器。

那么如何理解 CSS 盒子模式呢？在网页设计中，常用的属性有内容(Content)、填充(Padding)、边框(Border)、边距(Margin)等，CSS 盒子模式都具备这些属性。在日常生活中所见的盒子也具有这些属性。例如，"内容"就是盒子中装的东西；而"填充"就是怕东西损坏而添加的泡沫等抗震辅料，有的也叫"间隙"；"边框"就是盒子本身；"边距"则说明盒子摆放时彼此间要保留一定空隙。在网页设计上，"内容"常指文字、图片等元素。"填充"只有宽度属性，可以理解为生活中盒子里的抗震辅料厚度。"边框"有大小和颜色之分，可以理解为生活中盒子的厚度以及使用的颜色材料。边距就是该盒子与其他东西要保留多大距离。

假设在一个平面上，把不同大小和颜色的盒子，以一定的顺序和间隙摆放好，看到的就是如图 3-4 所示的一个盒子(Box)。盒子由里向外依次是 content、padding、border、margin。

CSS 将所有的 HTML 块元素看成是一个盒子，每个盒子都由以下部分组成：

- 内容：所有 HTML 元素的内容，如文本和图片。内容区域的宽和高用 width 和 height 属性来表示。
- 间隙：设置元素内容到元素边框的距离，即围绕 content 提供的空白。
- 边框：设定一个元素的边线。
- 边距：设置一个元素所占空间的边缘到相邻元素之间的距离。

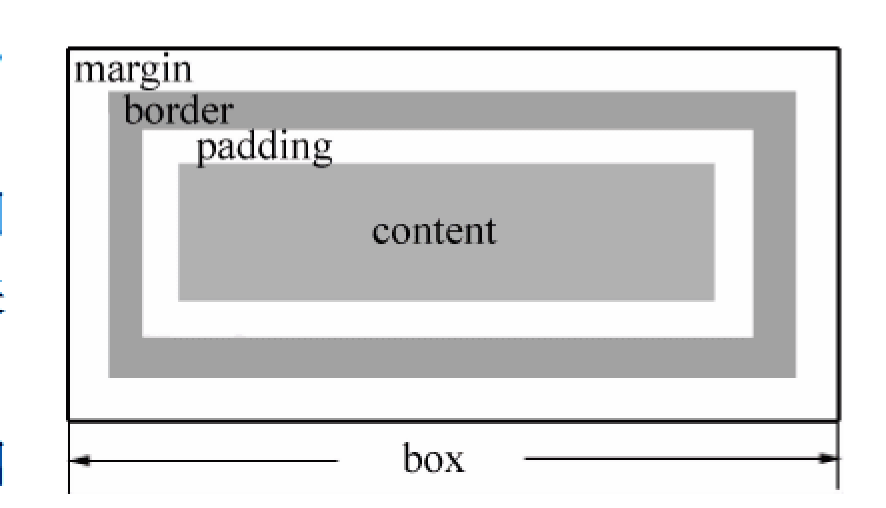


图 3-4　CSS 盒子模型

基于盒子模型，CSS 提供了很多手段以控制盒子的显示和排列方式。在网页中看到的"块"，多数对应于 HTML 的"块元素"，如<p>、<div>等。其中，<div>在网页布局中用的最多。

现在来看一个用<div>定义的、典型的版面分栏结构，即页头、导航栏、内容及版权，代码如下：

```
<body>
<div id = "header"></div>
<div id = "nav"></div>
<div id = "content"></div>
<div id = "footer"></div>
</body>
```

上述代码中，4 个用<div>定义的块就是盒子。希望这 4 个盒子等宽，并从上到下整齐排列，然后在整个页面居中对齐。为了方便控制，把这 4 个盒子装进一个更大的盒子<body>。要让最外边的大盒子在页面居中，并定义其宽度为 760 像素，同时加上边框，那么它的样式是：

```
body
{
font-family: Arial, Helvetica, sans-serif; font-size: 12px; margin: 0px auto;
height: auto; width: 760px; border: 1px solid #006633;
}
```

为了简单起见，页头的整个区块采用了一幅背景图，并在下边界设定间隙，这样使得页头的图像不和下面的导航栏连在一起。其样式代码为：

```
#header
{
height: 100px; width: 760px;
background-image: url (headPic.gif); background-repeat: no-repeat;
margin:0px 0px 3px 0px;
}
```

导航栏做成一个个小按钮，鼠标移上去会改变按钮背景色和字体色，这些小小按钮又可以理解为小盒子，如此一来就是一个盒子嵌套问题了，样式代码如下：

```
#nav {height: 25px; width: 760px; font-size: 14px; list-style-type: none; }
#nav li {float:left; }
#nav li a { color:#000000; text-decoration:none; padding-top:4px; display:block;
width:97px; height:22px; text-align:center;
background-color: #009966; margin-left:2px; }
#nav li a:hover { background-color:#006633; color:#FFFFFF; }
```

内容部分主要放入文章内容，有标题和段落，标题加粗，段落自动实现首行缩进 2 个字，同时所有内容要和外层大盒子边框有一定距离，这里用 padding。样式代码为：

```
#content {height:auto; width:740px; line-height:1.5em; padding:10px; }
#content p {text-indent:2em; }
#content h3 {font-size:16px; margin:10px; }
```

版权栏采用了一个纯色的背景与页头相映，里面文字自动居中对齐，有多行内容时，行间距合适。样式代码如下：

```
#footer
{
height:50px; width:740px; line-height:2em;
text-align:center; background-color:#009966; padding: 10px;
}
```

最后回到样式开头，大家会看到这样的样式代码：

```
*
{
margin: 0px; padding: 0px;
}
```

这是用了通配符初始化各标签的边距和间隙，就不用对每个标签加以这样的控制，在一定程度上简化了代码。

最终完成的 CSS 样式代码如下：

```
<style type="text/css">
<!--
* {margin: 0px;padding: 0px;}
body
{
font-family: Arial, Helvetica, sans-serif;font-size: 12px;margin: 0px auto;
height: auto;width: 760px;border: 1px solid #006633;
}
#header
{
height: 100px;width: 760px;background-image: url(headPic.gif);
background-repeat: no-repeat;margin:0px 0px 3px 0px;
}
#nav {height: 25px;width: 760px;font-size: 14px;list-style-type: none;}
#nav li {float:left;}
#nav li a{color:#000000;text-decoration:none;padding-top:4px;display:block;
width:97px;height:22px;text-align:center;background-color: #009966;margin-left:2px;
}
#nav li a:hover{background-color:#006633;color:#FFFFFF;}
#content {height:auto;width: 740px;line-height: 1.5em;padding: 10px;}
#content p {text-indent: 2em;}
#content h3 {font-size: 16px;margin: 10px;}
#footer {height: 50px;width: 740px;line-height: 2em;text-align: center;
background-color: #009966;padding: 10px;
}
-->
</style>
```

页面的结构代码如下：

```
<body>
<div id="header"></div>
<ul id="nav">
<li><a href="#">首页</a></li>
<li><a href="#">文章</a></li>
<li><a href="#">相册</a></li>
<li><a href="#">Blog</a></li>
<li><a href="#">论坛</a></li>
<li><a href="#">帮助</a></li>
</ul>
<div id="content">
<h3>前言</h3><p>第一段内容</p>
<h3>理解 CSS 盒子模式</h3><p>第二段内容</p>
</div>
<div id="footer">
<p>关于华升|广告服务|华升招聘|客服中心|QQ 留言|网站管理|会员登录|购物车</p>
```

```
<p>Copyright &copy;2006 - 2008 Tang Guohui. All Rights Reserved</p>
</div>
</body>
```

运行上述例子,得到的运行结果如图 3-5 所示。

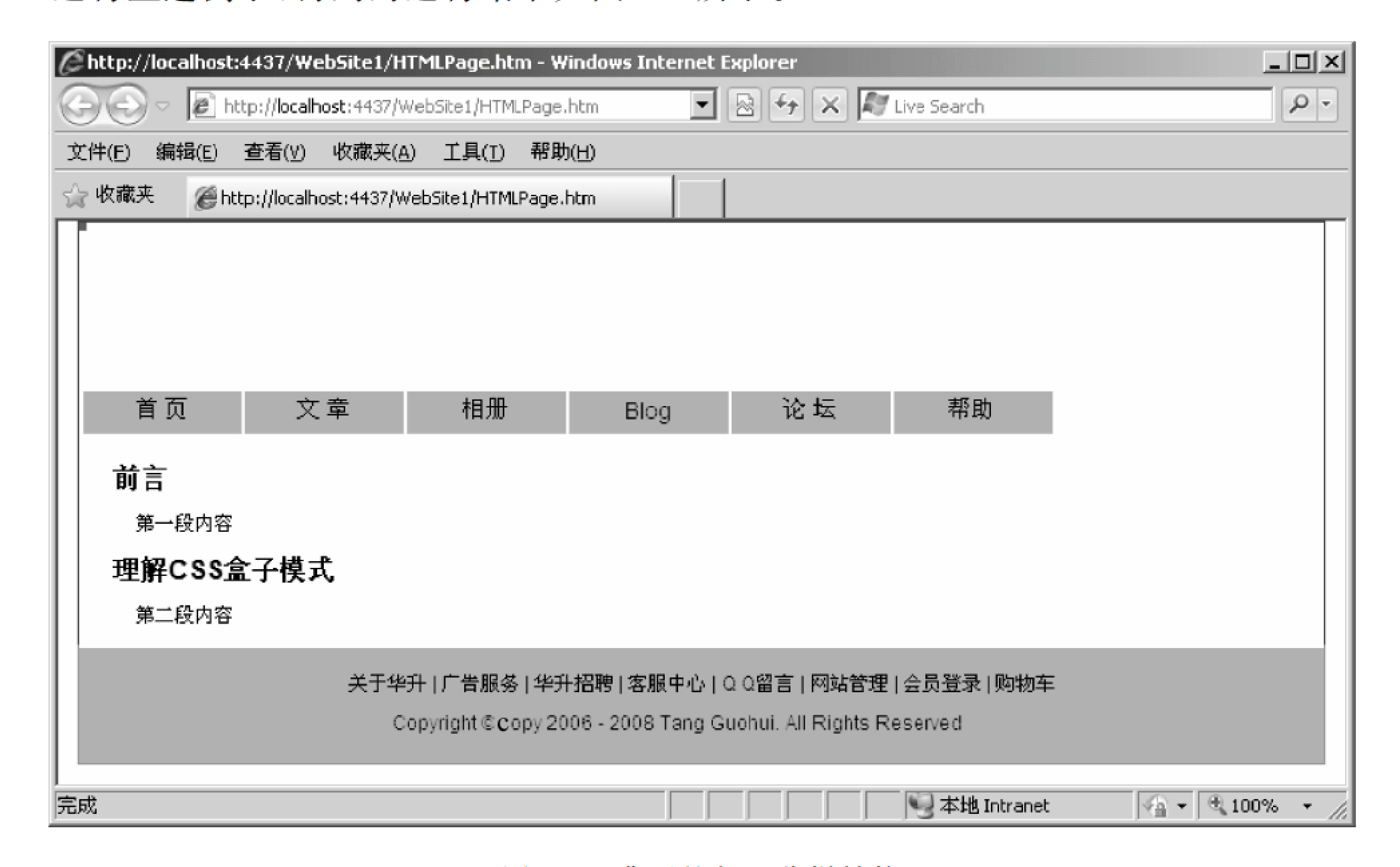


图 3-5 典型的版面分栏结构

3.6 习题与上机练习

1. 选择题

(1) 下面说法错误的是(　　)。

A. CSS 样式表可以将格式和结构分离

B. CSS 样式表可以控制页面的布局

C. CSS 样式表可以使许多网页同时更新

D. CSS 样式表不能制作体积更小下载更快的网页

(2) 若要在网页中插入样式表 main. css,以下用法正确的是(　　)。

A. <Link href="main. css" type=text/css rel=stylesheet>

B. <Link Src="main. css" type=text/css rel=stylesheet>

C. <Link href="main. css" type=text/css>

D. <Include href="main. css" type=text/css rel=stylesheet>

(3) 引用外部样式表的格式是(　　)。

A. <style src="mystyle. css">

B. <link rel="stylesheet" type="text/css" href="mystyle. css">

C. <stylesheet>mystyle. css</stylesheet>

(4) 引用外部样式表的元素应该放在(　　)。

A. HTML 文档的开始位置　　B. HTML 文档的结束位置

C. 在 head 元素中　　D. 在 body 元素中

(5) 下列(　　)是定义样式表的正确格式。

A. {body:color=black;}　　B. body:color=black

C. body {color: black}　　D. {body;color:black}

(6) 如果要在不同的网页中应用相同的样式表定义,应该(　　)。

A. 直接在 HTML 的元素中定义样式表

B. 在 HTML 的<head>标记中定义样式表

C. 通过一个外部样式表文件定义样式表

D. 以上都可以

(7) 样式表定义 #title {color:red} 表示(　　)。

A. 网页中的标题是红色的

B. 网页中某一个 id 为 title 的元素中的内容是红色的

C. 网页中元素名为 title 的内容是红色的

D. 以上任意一个都可以

(8) 下列(　　)是 css 正确的语法构成。

A. body:color=black　　B. {body;color:black}

C. body {color: black;}　　D. {body:color=black(body}

2. 案例分析题

(1) 解释以下 CSS 样式的含义。

```
table
{
    border: 1px #333 solid;
    font: 12px arial;
    width: 500px
}
td,th
{
        padding: 5px;
        border: 2px solid #EEE;
        border-bottom-color: #666;
        border-right-color:  #666;
}
```

(2) 解释以下 CSS 样式的含义。

```
form
{
      border:1px dotted #AAAAAA;
      padding:3px 6px 3px 6px;
      margin:0px;
      font:14px Arial;
}
```

```
select
{
    width:80px;
    background-color:#ADD8E6;
}
```

(3) 写出下列要求的 CSS 样式表。

① 设置页面背景图像为 login_back.gif,并且背景图像垂直平铺。

② 使用类选择器,设置按钮的样式,按钮背景图像为 login_submit.gif;字体颜色为 #FFFFFF;字体大小为 14px;字体粗细为 bold;按钮的边界、边框和填充均为 0px。

第4章 JavaScript 客户端编程

JavaScript 是一种解释型脚本语言,其最初设计目的是在 HTML 网页中增加动态效果和交互功能。随着 Web 技术的发展,JavaScript 与其他技术相结合,产生了客户端与服务器端异步通信的 AJAX 技术,为用户提供更加丰富的上网体验。本章主要讲解 JavaScript 语法及对象化编程等基础知识,后续的章节将专门讲解 AJAX 技术。

4.1 JavaScript 概述

4.1.1 什么是 JavaScript

JavaScript 最初由 Netscape 公司创造,起名 LiveScript,后来改名为 JavaScript。主要用于客户端脚本编程,通常嵌入在 HTML 网页中,由浏览器解释和运行。

JavaScript 与 Java 类似,都有分支、循环等控制结构及异常处理机制。还具有基于对象(Object Based)和事件驱动(Event Driven)的特性,可以通过文档对象模型(DOM)访问浏览器及页面中的各个对象,捕获对象的特定事件并编写代码处理事件。

JavaScript 可以实现的基本功能如下:

1. 控制文档的外观和内容

通过 Document 对象的 write 方法可以在浏览器解析文档时将 HTML 写入文档中;也可以在页面加载后,通过文档对像模型,找到页面中的某个元素动态改变其内容和外观。

2. 验证表单输入内容

在客户端验证表单中的输入,避免向服务器提交非法数据,节约服务器资源。

3. 实现客户端的计算和处理

直接从表单中读取客户端的输入,并进行相应计算。

4. 设置和检索 Cookie

将用户名、账号等用户的特定信息持久地保存于 Cookie 中,在用户下一次访问网站时,自动地读取这些信息,并根据这些信息实现个性化定制。

5. 捕捉用户事件并相应地调整页面

根据键盘或鼠标的动作,使页面的某一部分变得可编辑,或改变其显示样式。

6. 在不离开当前页面的情况下与服务器端应用程序进行交互

这是 AJAX 的基础,可以用于填充选项列表、更新数据以及刷新显示,并且不需要重新载入页面。

4.1.2 在网页中嵌入 JavaScript 脚本

与其他脚本语言一样,JavaScript 程序不能独立运行,只有把它嵌入到 HTML 网页中

才能运行。引入 JavaScript 脚本的方式有如下三种：

1. 在 HTML 文档中直接嵌入脚本程序

可以使用<script>标记，将 JavaScript 脚本块嵌入 HTML 页面中。用法如下：

```
<script type="text/javascript">
    JavaScript 脚本块;
</script>
```

JavaScript 脚本块可以放在 HTML 页面中的任何位置，但通常放在<head>标记内，因为这样在页面装载前脚本已经加载完成，页面中可以随时调用，若放在<body>内，则可能出现页面中调用脚本而脚本代码尚未加载从而出错的情况。

例 4-1 在 HTML 中嵌入 JavaScript 脚本(04-01. html)。

```
<html>
<head>
    <script type="text/javascript">
    document. write ("Hello, world!")    // 直接在页面中显示提示信息
    alert("Hello, world!")               // 弹出对话框显示提示信息
    </script>
</head>
<body>
</body>
</html>
```

上述代码中，document. write()是文档对象的输出函数，其功能是将括号中的内容输出到页面中；alert()是窗口对象的方法，用于弹出一个对话框。值得注意的是</script>中的代码区分大小写。例如，将 document. write()写成 Document. write()，程序将无法正确执行。

上述程序的运行结果如图 4-1 所示。



图 4-1 在 HTML 中嵌入 JavaScript 脚本

2. 在 HTML 文档中链接脚本文件

为了便于项目开发，通常将 JavaScript 代码保存到扩展名为 js 的文件中，这样这些代码就可以被多个 HTML 文件引用。在<script>标记中使用 src 属性可以导入外部脚本文件中的代码，格式如下：

```
<html>
    <script type="text/javascript" src="文件名.js"></script>
</html>
```

例 4-2 在 HTML 中链接外部脚本文件(04-02. html)。

```
<html>
    <head>
    <script src = "test01.js" type = "text/javascript"></script>
    </head>
    <body>
    </body>
</html>
```

其中,脚本文件 test01. js 的内容如下:

```
document.write("Hello, world!") ;
alert("Hello, world!");
```

该页面的显示效果与例 4-1 完全相同。

3. 在 HTML 标记内嵌入 JavaScript 代码

可以在 HTML 标记中嵌入 JavaScript 脚本代码,以便响应相关事件。例如,单击页面中的一个按钮,弹出 alert 对话框。

例 4-3 在 HTML 标记中嵌入 JavaScript 脚本并执行(04-03. html)。

```
<html>
<head><title>在标记内添加脚本测试</title></head>
<body>
    <button onClick = "alert('这是标记内的脚本!')">在标记内添加脚本测试</button>
</body>
</html>
```

在 button 标记的 onClick 属性中直接写了一句 JavaScript 代码,当单击该按钮时将触发执行该代码。

4.1.3 使用 JavaScript 输入与输出信息

在 JavaScript 中,常用的字符串输入输出方法有 document 对象的 write 方法、window 对象的 alert 方法、文本框及消息输入框等。消息框包括确认框、提示框等。

1. 利用 document 对象的 write 方法输出字符串

其功能是向页面输出文本,具体格式为:

```
document. write("待输出的字符串");
```

2. 利用 window 对象的 alert 方法输出字符串

其功能是弹出一个带“确定”按钮的对话框,并显示要输出的字符串,具体格式为:

```
window.alert ("待输出的字符串");
```

或简写为:

```
alert ("待输出的字符串");
```

3. 使用确认框

当需要确认或者接受某项操作时,通常使用 JavaScript 弹出一个确认框,用户必须单击

“确定”或“取消”按钮才能继续。

例 4-4 确认框示例(04-04.html)。

```
<html>
<head><title>确认框示例</title>
<script language="javascript">
    function test()
    {
        var value = confirm("确定要执行该操作吗?");
        alert("你的选择是: " + value);
    }
</script>
</head>
<body>
    确认框示例<button onClick="test()">测试</button>
</body>
</html>
```

当单击“测试”按钮时将调用 test 函数，从而弹出确认框，如图 4-2 所示。在确认框中单击“确定”按钮时，将返回 True，而单击“取消”按钮时将返回 False，根据返回值可以决定下一步的操作。

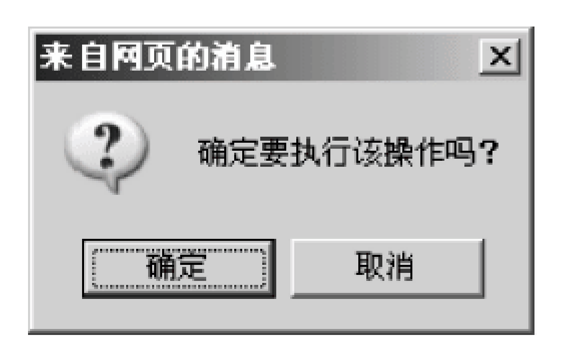


图 4-2 确认框示例

4. 使用提示输入框

程序中有时要弹出一个输入框，提示用户输入一段文本，这可以使用 window 对象的 prompt 方法来实现，格式如下：

```
prompt("提示文本","默认值")
```

例 4-5 提示输入框示例(04-05.html)。

```
<html>
<head>
<title>提示输入框示例</title>
<script language="javascript">
    function test()
    {
        var value = prompt("请输入你的名字","佚名")
        alert("您的名字是: " + value);
    }
</script>
</head>
<body>
  提示输入框示例<button onClick="test()">测试</button>
</body>
</html>
```

单击“测试”按钮，将弹出如图 4-3 所示的输入框。promt 方法的第一个参数代表提示信息，第二个参数代表默认的输入值。

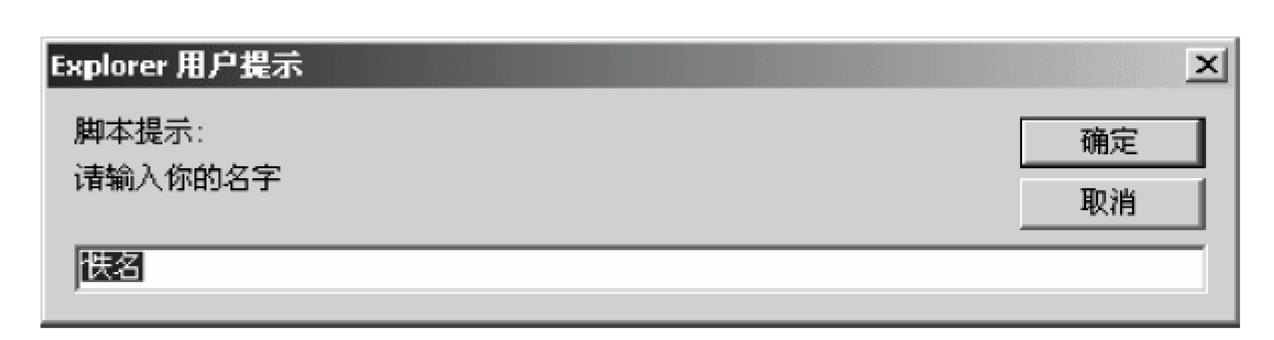


图 4-3 提示输入框示例

4.2 JavaScript 基本语法

同其他计算机语言一样，JavaScript 有它的基本数据类型、运算符、表达式及流程控制语句等。

4.2.1 数据类型

JavaScript 中可以使用如下 4 种基本数据类型：

- string(字符串)类型：是用单引号或双引号括起来的一个或几个字符。
- number(数值)类型：可以是整数或浮点数。
- boolean(布尔)类型：值为 true 或 false。
- object(对象)类型：用于定义对象。

4.2.2 变量

JavaScript 是一种弱类型语言，并不要求一定要对变量进行声明。为了避免混淆，最好养成声明变量的习惯。在 JavaScript 中，用关键字 var 来声明变量，语法如下：

```
var 变量名 1, 变量名 2, … , 变量名 n
```

声明中仅仅指定了变量名，在为变量赋值时系统会自动判断类型并进行转换。这也意味着在程序执行过程中，程序员可以根据需要随意改变某个变量的数据类型。

例如：

```
var test;                           // 声明变量
var level = 10, amount = 100;       // 变量声明的同时进行初始化.
test = 100;                         //为变量赋整数值
test = "Hello"                      //为变量赋字符串值
```

如果在声明时没有对变量进行初始化，变量将自动取值 undefined。

此外，JavaScript 还提供了强制类型转换函数，常用的有 Number 函数和 String 函数。

例如：

```
Number(ch);                      //将字符型数据"ch"转换为数值型
String(x);                       //将数值型数据 x 转换为字符型
```

变量名区分大小写，且必须符合如下的命名规则：

- 首字符可以是字母、美元符号($)以及下划线(_)，但不能是数字。
- 后续字符可以由字母、数字、下划线、美元符号组成。
- 不能使用 JavaScript 保留的关键字，如 var、for 等。

4.2.3 运算符和表达式

JavaScript 中，构成表达式的主要元素是运算符，根据运算符可以将表达式分为算术表

达式、关系表达式和逻辑表达式等，这些表达式可以共同构成一个复合表达式。

表4-1所示为将运算符按从高到低优先级进行排列的列表。

表4-1 JavaScript的运算符

<table>
<tr><th>描述</th><th>符　　号</th><th>说　　明</th></tr>
<tr><td>括号</td><td>(x) [x]</td><td>中括号只用于指明数组的下标</td></tr>
<tr><td rowspan="2">求反</td><td>－x</td><td>返回 x 的相反数</td></tr>
<tr><td>！ x</td><td>返回与 x (布尔值)相反的布尔值</td></tr>
<tr><td rowspan="2">自加</td><td>x＋＋</td><td>x 值加 1,但仍返回原来的 x 值</td></tr>
<tr><td>＋＋x</td><td>x 值加 1,返回后来的 x 值</td></tr>
<tr><td rowspan="2">自减</td><td>x－－</td><td>x 值减 1,但仍返回原来的 x 值</td></tr>
<tr><td>－－x</td><td>x 值减 1,返回后来的 x 值</td></tr>
<tr><td rowspan="5">算术
运算</td><td>x＊y</td><td>返回 x 乘以 y 的值</td></tr>
<tr><td>x/y</td><td>返回 x 除以 y 的值</td></tr>
<tr><td>x％y</td><td>返回 x 与 y 的模(x 除以 y 的余数)</td></tr>
<tr><td>x＋y</td><td>返回 x 加 y 的值</td></tr>
<tr><td>x－y</td><td>返回 x 减 y 的值</td></tr>
<tr><td>关系
运算</td><td>x＜y,x＞y
x＜＝y,x＞＝y
x＝＝y,x!＝y</td><td>符合条件时返回 true,否则返回 false</td></tr>
<tr><td rowspan="3">位运算</td><td>x＆y</td><td>位与：当两个数位同时为 1 时,返回 1,其他情况都为 0</td></tr>
<tr><td>x^y</td><td>位异或：两个数位中有且只有一个为 0 时,返回 0,否则返回 1</td></tr>
<tr><td>x|y</td><td>位或：x 或 y 为 1 则返回 1；当 x 和 y 均为 0 时返回 0</td></tr>
<tr><td rowspan="2">逻辑运算</td><td>x＆＆y</td><td>当 x 和 y 同时为 true 时返回 true,否则返回 false</td></tr>
<tr><td>x||y</td><td>当 x 和 y 任一个为 true 时返回 true；两者均为 false 时返回 false</td></tr>
<tr><td>条件运算</td><td>c? x:y</td><td>当条件 c 为 true 时返回 x,否则返回 y</td></tr>
<tr><td rowspan="6">赋值
运算</td><td>x＝y</td><td>把 y 的值赋给 x,返回所赋的值</td></tr>
<tr><td>x＋＝y</td><td>x 与 y 相加,将结果赋给 x,返回赋值后的 x 值</td></tr>
<tr><td>x－＝y</td><td>x 与 y 相减,将结果赋给 x,返回赋值后的 x 值</td></tr>
<tr><td>x＊＝y</td><td>x 与 y 相乘,将结果赋给 x,返回赋值后的 x 值</td></tr>
<tr><td>x/＝y</td><td>x 与 y 相除,将结果赋给 x,返回赋值后的 x 值</td></tr>
<tr><td>x％＝y</td><td>x 与 y 求余,将结果赋给 x,返回赋值后的 x 值</td></tr>
<tr><td>字符串
连接</td><td>x＋y</td><td>将 2 个字符串拼接起来当字符串与数字一起执行"＋"运算时,实际上也是执行连接运算。例如,x ＝ "5" ＋ 5,结果 x 的值为字符串"55"</td></tr>
</table>

位运算符通常被当做逻辑运算符来使用。它的实际运算情况是，把两个操作数(即 x 和 y)化成二进制数，对每个数位执行运算后，得到一个新的二进制数。通常，"真"值是全部数位为 1 的二进制数，而"假"值则全部数位为 0，所以位运算符可以充当逻辑运算符。

4.2.4 流程控制

JavaScript 提供了多种方式实现选择、循环等流程控制。

1. 选择结构

JavaScript 使用 if-else 语句或 switch 语句来实现选择结构的流程控制。

if-else 语句的格式如下：

```
If (条件表达式)
{
    语句体 1
}
else
{
    语句体 2
}
```

例如：

```
<script type="text/javascript">
    var d=new Date();
    var time=d.getHours();
    if (time<12)
        document.write("<b>Good morning</b>");
    else
            document.write("<b>Good afternoon</b>");
</script>
```

上述代码根据当前时间进行判断，然后在浏览器中显示相应的问候语。

JavaScript 的 if 语句也支持嵌套，其语法与 Java 完全一样，这里不再详细说明。

switch 语句用于多路选择控制，格式如下：

```
switch(expr)
{
    case 常量表达式 1: 代码段 1; break;
    case 常量表达式 2: 代码段 2; break;
    …
    case 常量表达式 n: 代码段 n; break;
    default: 默认代码段;
}
```

执行中，系统先对 switch 后面的 expr 求值，然后用该值与各 case 后的表达式值作比较。若与某 case 相匹配，则执行该 case 后面的代码段；若所有 case 表达式都不匹配，则执行 default 后的默认代码段。执行完一个代码段后，通常使用 break 语句跳出选择结构。

例 4-6 多路选择结构(switch 语句)示例 (04-06.html)。

```
<html>
<head>
  <title>switch 语句示例 </title>
  <script type="text/javascript">
     var now  = new Date();
     var date =  now.getDay();
     switch (date)
     {
        case 1:  alert("今天是星期一");    break;
        case 2:  alert("今天是星期二");    break;
```

```
            case 3:   alert("今天是星期三");     break;
            case 4:   alert("今天是星期四");     break;
            case 5:   alert("今天是星期五");     break;
            case 6:   alert("今天是星期六");     break;
            default:   alert("今天是星期日");
        }
    </script>
</head>
<body></body>
</html>
```

2. 循环结构

JavaScript 提供了如下三种循环控制语句：

1）while 语句

while 循环是当满足指定条件时，不断地重复执行循环体。语法格式如下：

```
while (条件表达式)
{
    循环体
}
```

例如，下面的程序实现 1～100 的累加。

```
sum = 0;
i = 1;
while(i <= 100)
{
    sum += i; i++;
}
```

2）do-while 语句

do-while 循环是 while 循环的一种变体，首先执行循环体，再判断条件表达式，如果条件表达式的值为真，则继续执行循环体，否则退出循环。也就是说，循环至少执行一次。其语法格式是：

```
do
{
    循环体
} while(条件表达式)
```

用 do-while 形式改写上面的累加和程序，代码如下：

```
sum = 0; i = 1;
do
{
    sum += i;   i++;
}
while(i <= 100);
```

3) for 语句

for 语句格式如下：

```
for( 循环变量赋初值; 循环条件; 循环变量增值)
{
    循环体
}
```

例如，改写上面的累加和程序如下：

```
sum = 0;
for(i=1; i<=100; i++) {
    sum += i;
}
```

同其他的程序设计语言一样，分支和循环都可以嵌套。请看如下的示例。

例 4-7　打印乘法口诀表(04-07.html)。

```
<html>
<head>
<title>for 循环语句打印乘法口诀</title>
<script type="text/javascript">
    for (var i = 1 ; i<= 9 ; i++)
    {
        for (var j = 1 ; j<= 9 ; j++)
        {
            if (j<=i)   //只打印下三角
            {
                document.write("\t"+j+"*"+i+"="+(i*j));
            }
        }
        document.write("<br/>");   //换行
    }
</script>
</head>
<body></body>
</html>
```

程序运行结果如图 4-4 所示。

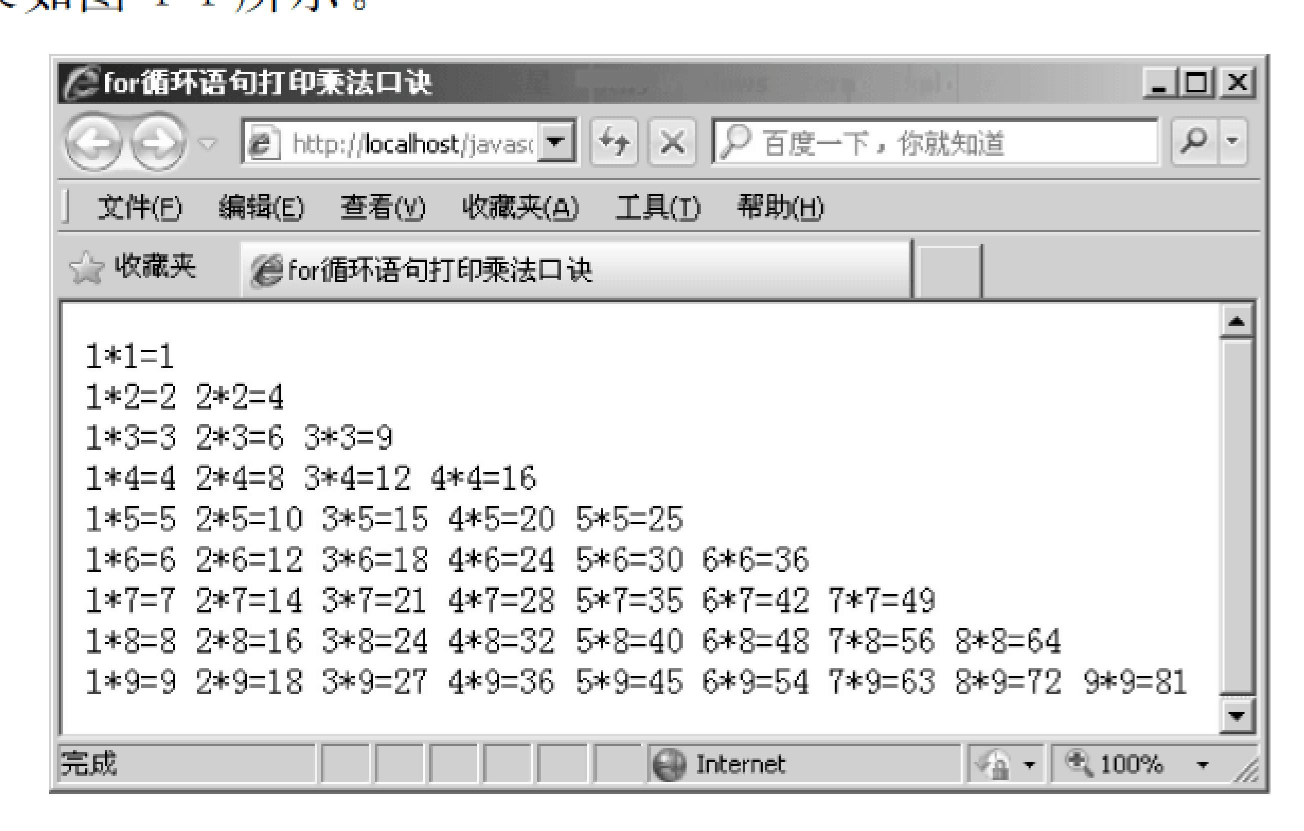


图 4-4　for 循环语句

4）break 语句和 continue 语句

在循环中经常用到 break 语句和 continue 语句，说明如下：

- break 语句：出现在循环语句块或 switch 语句块内，用于强行跳出循环或 switch。在嵌套循环中，break 语句只跳出当前循环体，并不跳出整个嵌套循环。
- continue 语句：用在循环结构中，作用是跳过循环体内剩余的语句而直接进入下一次循环。

例如：

```
sum = 0;
i = 0;
while(true)
{
   i + + ;
   if(i > 100)break;
   if(i % 2  =  = 0)continue;
   sum = sum + i;
}
```

上述程序的作用是求 1，2，…，100 中的奇数和。循环条件设置为永真，唯一能跳出循环的方法是当 i>100 时执行 break 语句。如果 i 为偶数，则执行 continue 语句跳过尚未执行的累加语句；如果 i 为奇数，则执行累加语句。

4.2.5 函数

函数是已命名的代码块，其中的语句作为一个整体被调用执行。

1. 定义函数

函数定义通常放在 HTML 文档的<head>块中，也可以放在其他位置，但要确保先定义后使用。JavaScript 中定义函数的一般形式如下：

```
function 函数名(形式参数表)
{
     语句块;
     return 返回值; // 无返回值的函数无此语句
}
```

说明：函数名是调用函数时所引用的名称，在同一个 JavaScript 脚本文件里函数名必须唯一。形式参数表用以接收传入数据，在调用函数时，其实参的个数和类型必须与形参相一致。大括号中是函数的执行语句，如果要返回一个值，则应该在最后一行使用 return 语句。

例如，下面的函数用于计算 n 的阶乘：

```
Function fact (n)
{
     var fact = 1;
     for (var i = 1; i < = n; i + + )      fact = fact * i;
     return fact;
}
```

2. 调用函数

有两种方式来调用函数：一是语句调用；二是事件调用。

1）语句调用

在程序语句中调用函数的形式如下：

```
函数名(实际参数表)
```

说明：实际参数应与定义函数时的形式参数一一对应，如果定义的时候没有参数，则调用的时候也不用参数，但括号不能省略。

例 4-8 函数的语句调用(04-08.html)。

```
<script type="text/javascript">
    a=2;
    b=3;
    add(a,b);
    function add(a,b)
    {
        alert("a与b两数之和为" + (a+b));
    }
</script>
```

当被调函数有返回值时，使用如下格式调用：

```
变量名 = 函数名(实际参数表)；
```

例 4-9 有返回值函数的调用(04-09.html)。

```
<script type="text/javascript">
    function add(a,b)
    {
        return (a+b);
    }
    var result = add(2,3);
    alert( "a与b两数之和为" + result);
</script>
```

2）事件调用

在网页中经常要捕获某些事件，由事件触发调用指定的函数。例如，当单击某按钮时调用某函数，或当鼠标指针指向某对象时调用某函数。

例 4-10 函数的事件调用(04-10.html)。

```
<html>
<head>
<title>javascript函数的事件调用</title>
<script type="text/javascript">
  function showmessage( )
  {
      alert("这是JavaScript事件调用函数");
```

```
  }
</script>
</head>
<body>
  <input type="button" value="鼠标单击事件调用函数" onClick="showmessage()" />
</body>
</html>
```

程序运行结果如图4-5所示。当用户单击页面中的按钮时，就会调用showmessage函数，弹出一个消息框。



图4-5 函数的事件调用

3. 变量的作用域

在函数之外定义的变量为全局变量，可在各个函数之间共享。在函数内部使用var声明的变量为局部变量，只在当前函数内部有效；但那些在函数内部没有用var声明的变量，在赋值后也会被当做全局变量使用。

例如：

```
function inc(n)
{
    y = ++n;
    return(y);
}
var x=3;
var sum=inc(x)+y;
alert(sum);          //sum的值是8
```

上述代码中，函数inc的内部没有用var声明变量y，当inc函数执行完后，局部变量y变成了一个全局变量，它的值仍然存在，所以inc(x)+y的值是8。但是，如果将inc(x)+y改成y+inc(x)，就会发生错误，因为y会先被引用到，此时y还没有被声明，所以产生错误。

4.2.6 异常处理

在程序运行过程中，引起错误发生通常有两种情况：一是程序内部的逻辑或语法错误；二是运行环境或用户输入了不可预知的数据。前者称为错误(Error)，可通过调试程序来解决；后者则常被称为异常(Exception)。因此，异常并不等价于错误；相反，有时还可利用异常来解决一些问题。JavaScript可以捕获异常并进行相应处理，从而避免了浏览器向用户报错。

1. 使用 try-catch-finally 处理异常

使用 try-catch-finally 结构处理可能发生异常的代码，格式如下：

```
try
{
    //要执行的代码
}
catch(e)
{
    //处理异常的代码
}
finally
{
    //无论异常发生与否,都会执行的代码
}
```

try 块用于捕获异常，当块中某一行代码抛出异常时，该行后面的代码将不再被执行，转而执行 catch 块的代码，在这里处理异常。若后面还有 finally 块，则不管 try 块中是否有异常产生，都要执行 finally 块中的语句。

catch 和 finally 块都是可以省略的，但至少要保留其中之一与 try 块结合使用。

在 catch 块中，括号中的参数 e 表示捕获到的异常对象实例，包含异常的详细信息，可以在这里根据不同的异常类型进行不同的处理。

finally 块中的语句始终会被执行，通常做一些最后的清理工作。如果在 try 块中遇到 return 等流程跳转语句，要跳出异常处理，程序的流程也会先执行 finally 中的代码，然后再进行跳转。

如果在一个异常处理语句中，只包含 try-finally 语句而没有处理异常的 catch 语句，则在 try 中抛出异常后会直接执行 finally 块的语句，最后再将异常向上抛出。

例 4-11　使用 try-catch-finally 处理异常(04-11.html)。

```
<html>
<head>
<title>使用 try-catch-finally 处理异常</title>
<script language="javascript">
    try
    {
        var date=new Date();
        date.test();      //调用 date 的未定义的 test 方法
        document.write("try 块执行结束<br>");
    }
    catch(error)
    {
        with(document)
        {
            write("出现了异常<br>");
            write("异常类型: "+error.name+"<br>");
            write("异常消息: "+error.message+"<br>");
        }
    }
    finally
    {
```

```
        document.write("<br>异常处理完毕!");
    }
</script>
</head>
<body></body>
</html>
```

程序运行结果如图 4-6 所示。



图 4-6 使用 try-catch-finally 处理异常

2. 使用 throw 语句抛出异常

前面的示例用 try-catch 结构处理了系统内置的异常类型，有时开发人员想抛出自定义的异常类型，以达到控制程序流并产生精确异常消息的目的，需要使用 throw 语句，基本格式如下：

```
Throw (exception);
```

Exception 就是要抛出的异常值，可以是字符串、整数、逻辑值或对象。

例 4-12 使用 throw 语句抛出异常示例(04-12.html)。

```
<script type="text/javascript">
  var x = prompt("Enter a number between 0 and 10:","")
  try
  {
    if(x>10)       throw "Err1"
    else if(x<0)   throw "Err2"
  }
  catch (er)
  {
    if(er= ="Err1")        alert("Error! The value is too high")
    else if(er = = "Err2") alert("Error! The value is too low")
  }
</script>
```

该代码块用于对一个输入的数值进行验证，若大于 10，则抛出字符串"Err1"，若小于 0，则抛出字符串"Err2"；在 catch 块中可以判断抛出的异常，并显示相应的提示信息。

4.2.7 JavaScript 事件处理

在客户端脚本中,JavaScript 通过事件响应来获得与用户的交互。例如,当用户单击一个按钮或在某段文字上移动鼠标时,就触发了一个单击事件或鼠标移动事件,通过对这些事件的响应,可以完成特定的功能(如单击按钮时弹出对话框,鼠标移动到文本上时文本变色等)。

1. 基本概念

JavaScript 是基于对象的语言,其基本特征就是事件驱动(Event Driven)。通常把鼠标或热键的动作称为事件(Event),把由事件引发的一连串程序的动作称为事件处理,对事件进行处理的程序或函数,被称为事件处理程序。

JavaScript 对事件的处理通常由函数完成,前面已经学习过如何由事件触发调用处理函数。通常浏览器会默认定义一些通用的事件处理程序,以便响应那些最基本的事件。例如,单击超链接的默认响应就是装入并显示目标页面,单击表单提交按钮的默认响应就是将表单提交到服务器等。虽然如此,要实现动态的、具有交互功能的页面,经常要自定义事件处理函数,这样可以让页面完成定制的处理功能。

2. JavaScript 标准事件

JavaScript 针对文档、表单、图像、超链接等对象定义了若干个标准事件,同时针对常用的 HTML 标记定义了事件处理属性,以便指定事件处理代码。下面简要介绍一些常用的 JavaScript 事件。

1) onload 和 onUnload 事件

当用户进入页面时,会触发 onload 事件;当退出一个页面时,会触发 onUnload 事件。若在 body 标记的 onload 或 onUnload 属性中设定了事件处理程序,则页面加载和退出时会自动执行该程序代码。请看如下代码:

```
<body onLoad = "alert('Welcome to JavaScriptworld! ');" >
    // 页面代码
</body>
```

这样,每次进入该页面时都会自动弹出 Welcome to JavaScriptworld! 消息框。

2) onClick 事件

当用户单击按钮或超链接时,就触发了 onClick 事件,由 onClick 属性指定的事件处理程序将被调用。例如:

```
<button name="button1" onClick = "btn1Click();"> click me</button>
```

这样,当用户单击该按钮时,会自动调用 btn1Click 函数。

3) onFocus、onBlur 和 onChange 事件

这三个事件通常与输入元素(text、textarea 及 select 等)配合使用。当某元素获得焦点时触发 onFocus 事件,当元素失去焦点时将触发 onBlur 事件,当元素失去焦点且内容被改变时,将触发 onChange 事件。这三个事件经常配合使用来验证表单输入的内容。

例如:

```
<input type="text" id="email" onchange="checkEmail()" />
```

这样，当 email 输入框的值改变时，会自动调用 checkEmail 函数来验证输入是否合法。

4）onMouseOver 和 onMouseOut 事件

当鼠标移向某个对象时将触发 onMouseOver 事件，当鼠标移出某个对象时将触发 onMouseOut 事件，这两个事件通常用来为页面对象创建一些动态效果。请看如下代码：

```
<a href="#" onmouseover="alert('An onMouseOver event');return false">Click Me</a>
```

当鼠标指向超链接时，就会弹出一个消息框。

5）onSubmit 事件

当表单提交时，会触发该事件。经常要在表单提交以前验证所有的输入域，以保证数据的正确性，这时就可以使用 onSubmit 事件。

请看如下代码：

```
<form method="post" action="xxx.aspx" onsubmit="return checkForm()">
    // 表单内容
</form>
```

当用户单击表单中的确认按钮时，checkForm 函数就会被调用。假若域的值无效，此次提交就会被取消。checkForm 函数的返回值是 True 或者 False。如果返回值为 True，则提交表单，反之取消提交。

关于更多 JavaScript 事件驱动的知识，读者可查阅相关书籍。

4.3 JavaScript 对象编程

面向对象技术是当前软件开发的主流方向，JavaScript 也支持面向对象编程。在 JavaScript 中可以定义类，并创建对象实例，也可以使用 JavaScript 内建的类和对象，还可以访问浏览器及文档对象模型中的对象。可以说，JavaScript 为对象化编程提供了强大的支持。

JavaScript 对象可以是一段文字、一幅图片、一个表单(Form)等，可以从属性、方法两个方面来描述对象。属性反映对象某些特定的性质，如字符串的长度、图像的长宽、文本框(Textbox)里的文字等；方法指对象可以执行的行为(或可以完成的功能)，如 String 对象的 toUpperCase 方法可以将所有字符转换为大写。要引用对象的某一“性质”，应使用“＜对象名＞.＜性质名＞”这种写法。

本节主要介绍 JavaScript 内置对象的使用，以及浏览器对象、文档对象模型中的对象使用；关于自定义类和对象的方法，请参阅相关书籍。

4.3.1 常用 JavaScript 对象

1. String 对象

字符串是 JavaScript 的一种基本的数据类型，声明一个 String 对象的最简单方法就是

直接赋值。

例如：

```
var s = "JavaScript"
```

String 类只有一个常用属性，即 Length 属性，返回字符串长度。

String 类的常用方法如表 4-2 所示。

表 4-2 String 类的常用方法

方 法	描 述
charAt()	返回在指定位置的字符
concat()	连接字符串
indexOf()	检索字符串
lastIndexOf()	从后向前搜索字符串
match()	找到一个或多个正则表达式的匹配
replace()	替换与正则表达式匹配的子串
search()	检索与正则表达式相匹配的值
split()	把字符串分割为字符串数组
substr()	从起始索引引号提取字符串中指定数目的字符
toLowerCase()	把字符串转换为小写
toUpperCase()	把字符串转换为大写

请看如下代码：

```
var msg = "Hello" + "World" ;                // "+"用于字符串,可以实现字符串拼接
var msg = concat("Hello", "World");          // 和上句等效
document.writeln( msg.length );              // 输出字符串的长度 10
var idx = msg.indexOf("World");              // 在字符串中检索子串出现的位置,这里为 5
// 截取从第 5 个字符往后到第 10 个字符之间的所有字符,为 World
document.writeln( msg.substring(idx,10) );
document.writeln(msg.toUpperCase());         // 输出为"HELLOWORLD"
```

2. Array 对象

Array 为数组对象，可以在单个变量中存储多个值。

通常使用如下方法来创建和访问数组：

```
var mycars = new Array();                          // 创建数组对象,可以不用指定元素个数
mycars[0] = "BMW";                                 // 为数组元素赋值
mycars[1] = "AUDI";
var yourcars = new Array(3);                       // 创建数组对象并指定元素个数
var hiscars = new Array("Buick", "Benz", "Volvo"); // 创建数组对象并同时赋值
for (x in mycars)                                  // 遍历数组元素,x 能得到元素下标
{
    document.write(mycars[x] + "<br />")           // 输出数组元素
}
```

同样，数组对象也有 length 属性，用于设置或返回数组中的元素个数。

数组的常用方法如表 4-3 所示。

表 4-3　数组的常用方法

方　法	描　述
concat()	连接两个或更多的数组，并返回结果
join()	把数组的所有元素放入一个字符串。元素通过指定的分隔符进行分隔
sort()	对数组的元素进行排序
reverse()	颠倒数组中元素的顺序
shift()	删除并返回数组的第一个元素
pop()	删除并返回数组的最后一个元素

例如：

```
var arr1 = new Array("Tom", "Jerry")
var arr2 = new Array("Bingo")
var arr = arr1.concat(arr2);           // 连接两个数组，生成新的数组
document.write(arr.join(" | "));       // 将 arr 中的元素拼接成字符串，使用"|"分隔
arr = new Array(7, 5, 3, 8, 6);
document.write(arr.sort());            // 对数组元素进行排序
```

3. Date 对象

Date 对象用于表示日期和时间，通过它可以进行一系列与日期、时间有关的操作。

可以使用如下方法创建 Date 对象，并为其赋以日期和时间值。

```
var d = new Date();                        //创建日期对象
d.setFullYear(2011,11,1);                  //赋日期值
d.setHours(9,58,58,0);                     //赋时间值
document.write(d.getYear() + "-" + d.getMonth() + "-" + d.getDate());  //输出日期
document.write("<br>" + d.toLocaleDateString());  // 显示为 2011 年 12 月 1 日 星期四
```

需要注意的是，当为 Date 对象设置日期时，月份可接收的数值为 0～11，代表 1～12 月；所以这里设置月份值为 11 时，实际上作为 12 月处理。

Date 对象的常用方法如表 4-4 所示。

表 4-4　Date 对象的常用方法

方　法	描　述
Date()	返回当日的日期和时间
getDate()	从 Date 对象返回一个月中的某一天(1～31)
getDay()	从 Date 对象返回一周中的某一天(0～6)
getMonth()	从 Date 对象返回月份(0～11)
getFullYear()	从 Date 对象以四位数字返回年份
getHours()	返回 Date 对象的小时(0～23)
getMinutes()	返回 Date 对象的分钟(0～59)
getSeconds()	返回 Date 对象的秒数(0～59)
setDate()	设置 Date 对象中月的某一天(1～31)
setMonth()	设置 Date 对象中月份(0～11)

续表

方　　法	描　　述
setFullYear()	设置 Date 对象中的年份(4 位数字)
setHours()	设置 Date 对象中的小时(0～23)
setMinutes()	设置 Date 对象中的分钟(0～59)
setSeconds()	设置 Date 对象中的秒钟(0～59)
setTime()	以毫秒设置 Date 对象

若只是创建 Data 对象而不为其赋值,则可从中获得当前的日期和时间。

例 4-13　Date 对象使用(时钟显示)(04-13.html)。

```
<html>
<head>
<script type="text/javascript">
    function startTime()
    {
      var d = new Date();
      var h = d.getHours();              // 获取当前的时、分、秒
      var m = d.getMinutes();
      var s = d.getSeconds();
      m = checkTime(m);                  // 若分、秒的值小于 10,则在前面补 0
      s = checkTime(s);
      // 在 div 上显示当前时间
      document.getElementById('clock').innerHTML = "本地时间: " + h+":"+m+":"+s;
      setTimeout( 'startTime()', 500);  // 设置 500 毫秒后再次调用该函数以更新时间
    }
    function checkTime(i)
    {
      if(i<10)   i = "0" + i;
      return i
    }
</script>
</head>
<body onload="startTime()">
    <div id="clock" />
</body>
</html>
```

4. Math 对象

Math 对象用于执行一些数学计算任务,该对象不需创建,可以直接使用。

JavaScript 提供了 8 个可被 Math 对象访问的算术值,如自然对数 e、圆周率 π 等。可以通过 Math 对象的属性访问,如 Math.E、Math.PI。

另外,调用 Math 对象的方法可以实现一些常用的数学计算。

例如:

```
var pi_val = Math.PI;                    // 获得 PI 值
var sin_val = Math.sin(pi_val);          // 计算 sin(PI)
var sqrt_val = Math.sqrt(5);             // 求 5 的平方根
```

```
var rnd_val = Math.round(sqrt_val);           // 对5的平方根取整
document.write("PI = " + pi_val + "<br/>");
document.write("sin(PI) = " + sin_val + "<br/>");
document.write("sqrt(5) = " + sqrt_val + "<br/>");
document.write("round(sqrt(5)) = " + rnd_val + "<br/>");
```

4.3.2 浏览器宿主对象

浏览器作为 JavaScript 的运行环境,提供了一系列的宿主对象;通过这些对象,JavaScript 可以获取浏览器的信息,并控制浏览器执行指定的操作。这些对象包括 window、navigator、screen、history、location、document 等。它们的关系如图 4-7 所示。

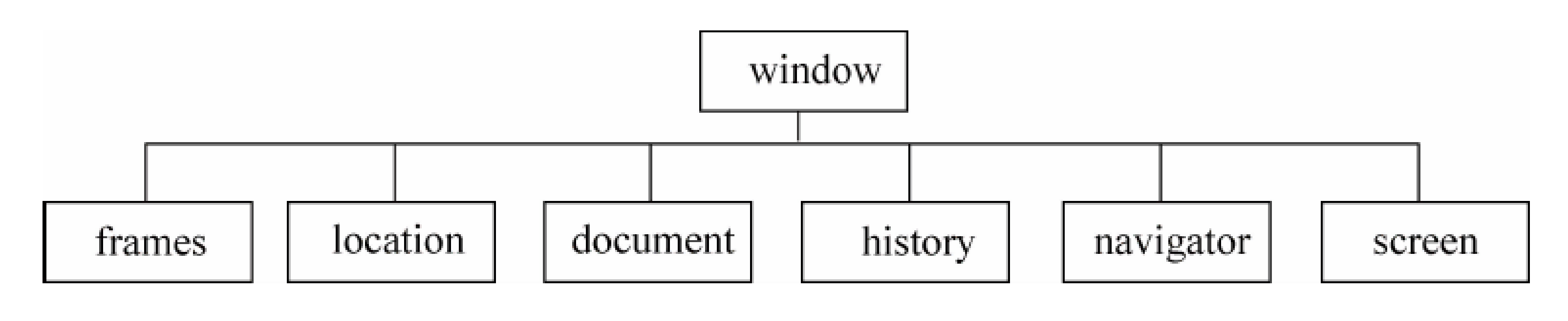


图 4-7 浏览器宿主对象关系

可以看出,window 是一个顶层的对象,其他对象都包含在 window 内部,通过 window 可以访问到其他对象;document 是最重要的一个对象,包含了很多与 HTML 元素相关的成员,可以访问加载到浏览器中的 HTML 文档,并且可以实现动态控制。

1. window 对象

window 对象表示浏览器中打开的窗口。如果文档包含框架(frame 或 iframe),那么浏览器会为 HTML 文档创建一个 window 对象,并为每个框架创建一个额外的 window 对象。

window 作为顶层对象,在访问它的属性和方法时,一般无须指定对象名。例如,下面两个调用是完全等价的:

```
window.alert( " 欢迎进入 JavaScript!") ;
alert( " 欢迎进入 JavaScript!") ;
```

使用 window 提供的 alert、confirm、prompt 等方法可以完成基本的浏览器交互,这些内容在前面已经学过,这里不再重复讲解。

使用 window 对象的 open 方法,可以打开一个新的窗口,语法格式如下:

```
window.open( [sURL] [, sname] [,sfeatures]) ;
```

说明:

sURL:打开网页的 url 地址,若该参数缺省,则打开空白网页。

sname:被打开窗口的名称,可以使用_top、_blank、_parent、_self 等内建名称,也可以自定义一个名称,以后可以使用该名称引用该窗口。

sfeatures:指定被打开窗口的特征,如窗口的宽度、高度、是否需要菜单条等,若要打开一个普通窗口,可以忽略该参数。

例如，下面的代码将打开一个 300×200 的空白窗口，并且没有菜单条和工具条。

```
window.open( " ", "_blank", "width = 300, height = 200, menubar = no, toolbar = no") ;
```

又如，下面的代码将在顶层框架中打开 163 邮箱首页。

```
window.open( " mail.163.com", "_top" ) ;
```

调用 window 对象的 close 方法可以关闭一个窗口，代码如下：

```
window.close() ;
```

打开一个窗口后，你可以设置或移动窗口的位置。

例如：

```
window.moveTo( 300, 200 ) ;     // 绝对定位方法,以屏幕左上角为原点
```

或

```
window.moveBy( - 10, - 10 ) ;     // 相对定位方法,以当前窗口左上角为原点
```

moveTo 方法以屏幕左上角为坐标原点，将窗口移动到指定位置，两个偏移量都必须为正；moveBy 方法则以当前窗口左上角为坐标原点，实现相对偏移，若偏移量为正，则向右、下方向移动，若为负，则向左、上方向移动。

使用 resizeTo 方法可以调整窗口的大小。

例如：

```
var w = window.screen.availwidth / 2 ;
var h = window.screen.availheight / 2 ;
window.resizeTo( w, h ) ;       // 将窗口的宽度和高度调整到屏幕宽、高的一半
```

在 JavaScript 中，有时需要以指定的时间间隔反复调用某函数，可使用 window 对象的 setInterval 方法实现。

例如：

```
var intervalID = window.setInterval( myfunction, 1000 ) ;
```

这将每隔 1 秒钟(1000 毫秒)自动调用一次名为 myfunction 的函数。若要取消该间隔调用，可使用 clearInterval 方法，如下：

```
window.clearInterval( intervalID ) ;
```

有时用户希望窗体加载后延迟一段时间，然后执行某项操作，这可通过调用 window 对象的 setTimeout 方法来实现。

例如：

```
var timeoutID = window.setTimeout( myfunction, 1000 ) ;
```

这将在1秒钟后自动调用myfunction方法。同理,使用clearTimeout方法可以取消延迟调用:

```
window.clearTimerout( timeoutID ) ;
```

除上面介绍的方法外,在程序中还经常访问window对象的一些属性,如表4-5所示。

表4-5　window对象的常用属性

属　　性	描　　述
document, screen, history, location, navigator 等	引用几个下级对象
frames	集合对象,代表当前窗口中的框架集,从而可以获取并操纵所有的子窗口
opener	代表使用open方法打开当前窗口的窗口
self	代表当前窗口
top	代表所有框架中的顶层窗口
status	代表窗口的状态栏
XMLHttpRequest	同服务器端异步交互的对象

例如:

```
window.location.href = "http://cn.yahoo.com";        // 当前窗口跳转到yahoo主页
window.status = "欢迎使用本系统";                       // 在窗口状态栏显示欢迎信息
```

2. location对象

location对象描述的是浏览器所打开的网页地址。要表示当前窗口地址,直接使用location或window.location即可,若要表示指定窗口的地址,则使用"窗口名.location"的格式。

例如:

```
var newwin = window.open("http://localhost/login.htm", "_blank") ;
document.write( newwin.location ) ;
```

使用location对象的属性可以获取详细的地址信息。

例如:

```
document.write("当前位置: " + location.href + "<br/>");
document.write("主机名称: " + location.host + "<br/>");
document.write("请求路径: " + location.pathname + "<br/>");
document.write("主机端口: " + location.port + "<br/>");
document.write("请求字符串: " + location.search + "<br/>");
```

location对象的常用方法如表4-6所示。

表 4-6 location 对象的常用方法

方　　法	描　　述
assign	加载一个新的 HTML 文档
reload	刷新当前网页，相当单击浏览器的“刷新”按钮
replace	打开一个新的 URL，并取代历史中的 URL

这三个方法的使用格式非常简单。

例如：

```
location.assign("http://localhost/login.aspx");
location.replace("http://localhost/index.html");
location.reload();
```

3. history 对象

history 对象代表了浏览器的浏览历史。鉴于安全性考虑，该对象的使用受到了很多限制，目前只能使用 back、forward 和 go 等几个方法，格式如下：

```
history.back( [num] )          // 浏览器后退 n 步
history.forward()              // 浏览器前进 1 步
history.go(location)           // 浏览器跳转到指定的网页
```

在 go 方法中，location 可以是一个 URL 字符串，也可以是一个整数。若是字符串，则代表了历史列表中的某个 URL；若是整数，则代表前进（正数）或后退（负数）的步数。若 location 为 0，则刷新当前页面，等同于 location. reload( )调用。

4.3.3 DOM 对象

当一个 HTML 网页被加载到浏览器中，浏览器会首先解析该网页，将其转换为文档对象模型(Document Object Model，DOM)，然后在内存中处理模型中的各个对象，最后将结果展示给用户。

文档对象模型是 HTML 文档在内存中的表示形式，定义了文档的逻辑结构，以及一套访问和处理文档的方法。浏览器是一个处理 HTML 文档的应用程序，必须将文档解析为 DOM 才能够以编程方式读取、操作和显示 HTML 文档。有了 DOM，JavaScript 就可以方便地访问页面元素，动态地改变页面的外观和行为。

在 DOM 中，文档的逻辑结构可以用节点树的形式表述，如图 4-8 所示。每个文档必须有一个 document 节点（对应于 html 元素），作为树的根节点，其他元素都是它的子节点。

1. 使用 document 对象

document 对象代表窗口中显示的文档，使用 window 对象的 document 属性即可返回一个 document 对象。

例如：

```
window.document        // 获得当前窗口的 document 对象
thewindow.document     // 获得指定窗口中的 document 对象
```

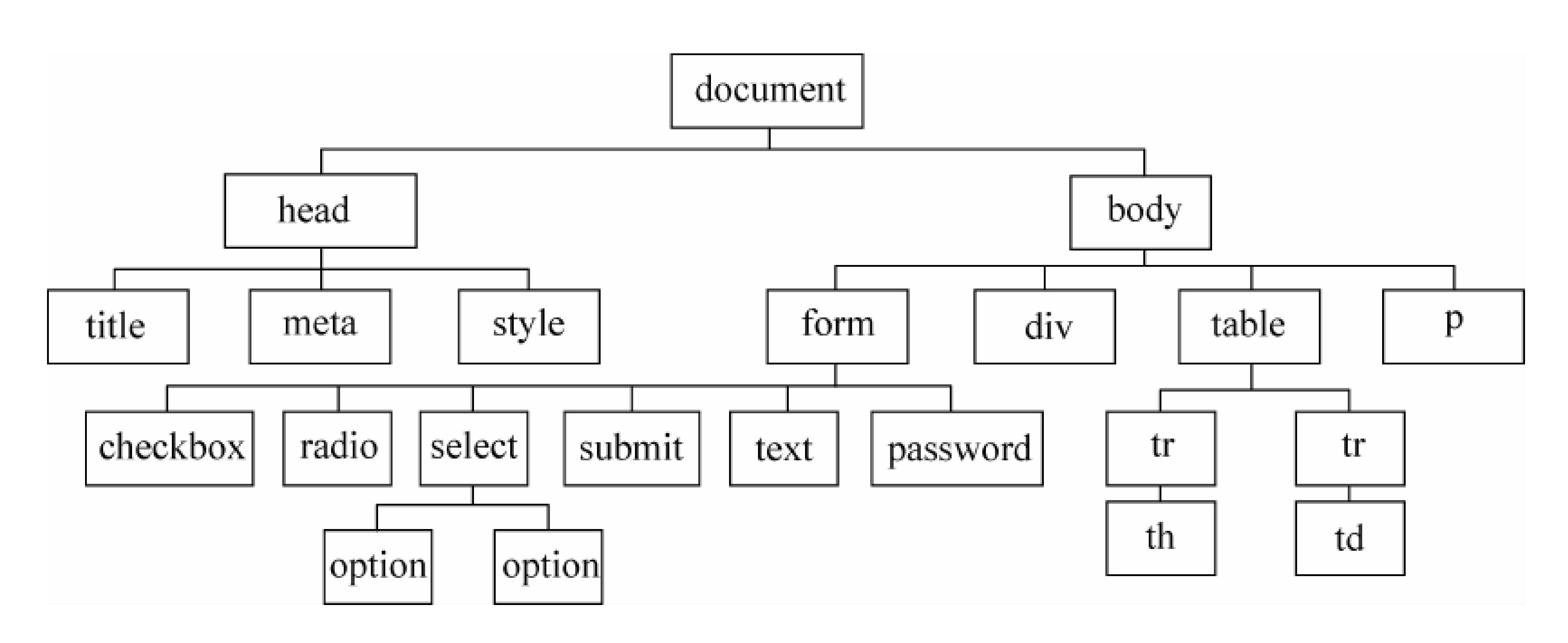


图 4-8　用节点树的形式表述文档的逻辑结构

通常使用 document 对象完成如下操作：

(1) 开启新的文档。

(2) 向网页中动态写入内容。

(3) 通过 document 获取对其他页面元素的引用，进而操作这些对象。

调用 document.open 方法可以打开一个新窗口，并加载一个新的文档，其用法与 window.open 方法基本类似。

使用 document.write 方法或 document.writeln 方法可以将指定的字符串写到当前文档中去，字符串中可以包含 HTML 标签，甚至包含 JavaScript 代码等。这两个方法的区别是，后者会在写完数据后再加一个回车符，因此下面的两个输出语句作用等价：

```
document.writeln( "Hello" ) ;
document.write( "Hello" + "\r" ) ;
```

由于在 HTML 中，回车符通常会被忽略，所以两条语句的输出结果差异不大。

2. 处理文档中的元素

在 JavaScript 中，经常要获得文档中的某个 HTML 元素，并动态改变其显示内容。有了 document 对象，就可以使用多种方式获得对页面元素的引用。

document 对象具有如表 4-7 所示的集合属性，可以获得指定元素的集合。从集合中再根据元素索引即可定位到具体的元素。

表 4-7　document 对象具有的集合属性

集　　合	描　　述
anchors[]	返回对文档中所有 Anchor 对象的引用
forms[]	返回对文档中所有 Form 对象的引用
images[]	返回对文档中所有 Image 对象的引用
links[]	返回对文档中所有 Link 对象的引用

例如，使用下面的代码可以获取并显示页面上图像的数量。

```
document.write("本文档包含: " + document.images.length + " 幅图像.")
```

下面的代码可以动态改变超链接元素的显示文本和链接目标等属性。

```
<html>
<head>
<script type="text/javascript">
    function change()
    {
        var ah = document.anchors[0];
        ah.innerHTML = "新的链接";
        ah.href = "http://mail.163.com";
    }
</script>
</head>
<body>
    <a href="#">旧的链接</a>
    <input type="button" onclick="change()" value="测试" />
</body>
</html>
```

要访问表单中的元素，通常使用如下格式：

```
document.表单名.元素名
```

请看如下的代码：

```
<html>
<head>
<script type="text/javascript">
    function test()
    {
        var name = document.form1.tbxname.value;
        var gd = document.form1.rbngd[0].checked?"男":"女";
        alert("你的输入是: " + name  + "\t" + gd);
    }
</script>
</head>
<body>
<form action="" name="form1">
姓名: <input name="tbxname" type="text" /><br/>
性别: <input name="rbngd" type="radio" />男
<input name="rbngd" type="radio" />女
<input type="button" name="btntest" onclick="test()" value="测试"  />
</form>
</body>
</html>
```

这里，使用 document.form1.tbxname 获得了对姓名输入框的引用，并通过 value 属性获得了其输入值；使用 document.form1.rbngd 将获得对单选钮的引用，由于有多个同名的单选钮存在，所以会返回一个元素集合，然后使用下标来指定不同的元素。

在新的 DOM 标准中，推荐使用 getElementById()和 getElementsByName()获取对页面元素的引用。前者要求被访问的 HTML 元素必须具有唯一的 ID，这样调用该方法将返回唯一的一个元素；后者将根据元素的名称进行检索，由于允许多个元素同名，所以该方法将返回一个元素的集合，同样使用下标来确定具体的元素。

例 4-14 使用 getElementById()和 getElementsByName()检索页面元素(04-14.html)。

```
<html>
<head>
<script type="text/javascript">
    function test()
    {
        var name = document.getElementById("tbxname").value;
        var favs = document.getElementsByName("cbxfav");
        var fstr = name + " 爱好:<br/>";
        for(var i=0; i<favs.length; i++ )
        {
            if(favs[i].checked==true)
            {
                fstr += favs[i].value;
                fstr += "<br />";
            }
        }
        document.getElementById("divmsg").innerHTML = fstr;
    }
</script>
</head>
<body>
姓名:<input id="tbxname" type="text" /><br/>
爱好:<input type="checkbox" name="cbxfav" value="唱歌" />唱歌
<input type="checkbox" name="cbxfav" value="跳舞" />跳舞
<input type="checkbox" name="cbxfav" value="体育" />体育
<input type="button" onclick="test()" value="测试" />
<div id="divmsg" />
</body>
</html>
```

程序的运行结果如图 4-9 所示。



图 4-9 例 4-14 的运行结果

在 HTML DOM 模型中，各种 HTML 元素都被当做对象使用，关于各种 DOM 对象的属性和方法，请参阅 DOM 文档。

在 JavaScript 中，不但可以访问 DOM 对象，还可以动态的创建或删除 DOM 对象，从而动态改变页面内容，请看如下示例：

```
var newoption = document.createElement("option");  // 创建一个 option 元素
newoption.innerHTML = "NewItem";                    // 设置 option 元素属性
newoption.value = 4;
// 将新建的 option 加入到页面上得 Select 对象中
document. getElementById("mySelect").appendChild(newoption) ;
// 删除 Select 对象中的第 0 个元素
document. getElementById("mySelect").remove(0) ;
```

4.4 JavaScript 编程实例

4.4.1 表单提交验证

在应用系统开发中，经常会遇到对表单内容进行验证的要求。例如，在用户注册时，需要对填写的用户名及密码进行验证。

例 4-15 表单提交验证示例(04-15. html)。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=GB2312" />
<script type="text/javascript">
function validate()
{
  var at=document.getElementById("email").value.indexOf("@") ;
  var age=document.getElementById("age").value ;
  var fname=document.getElementById("fname").value ;
  submitOK="true" ;
  if(fname.length==0||fname.length>10)
  {
    alert("姓名必须输入 1～10 个字符") ;
    submitOK="false" ;
  }
      else if (isNaN(age)||age<1||age>100)
  {
    alert("年龄必须是 1～100 之间的数字") ;
    submitOK="false" ;
  }
  else if (at==-1)
  {
    alert("不是有效的电子邮件地址") ;
    submitOK="false" ;
  }
```

```
    if (submitOK = = "false")
    {
      return false ;
    }
  }
</script>
</head>
<body>
<form action = "submitpage.html" onsubmit = "return validate()">
    姓名：<input type = "text" id = "fname" size = "20"><br />
    年龄：<input type = "text" id = "age" size = "20"><br />
    电邮：<input type = "text" id = "email" size = "20"><br />
    <input type = "submit" value = "提交">
</form>
</body>
</html>
```

程序运行后，如果什么都不输入就直接单击“提交”按钮，则看到如图 4-10 所示的结果。

当姓名输入空值或超过 10 个字符时，会提示“姓名必须输入 1～10 个字符”；当年龄输入 1～100 之外的值时提示“年龄必须是 1～100 之间的数字”；当电邮中没有@符号时，提示“不是有效的电子邮件地址”。

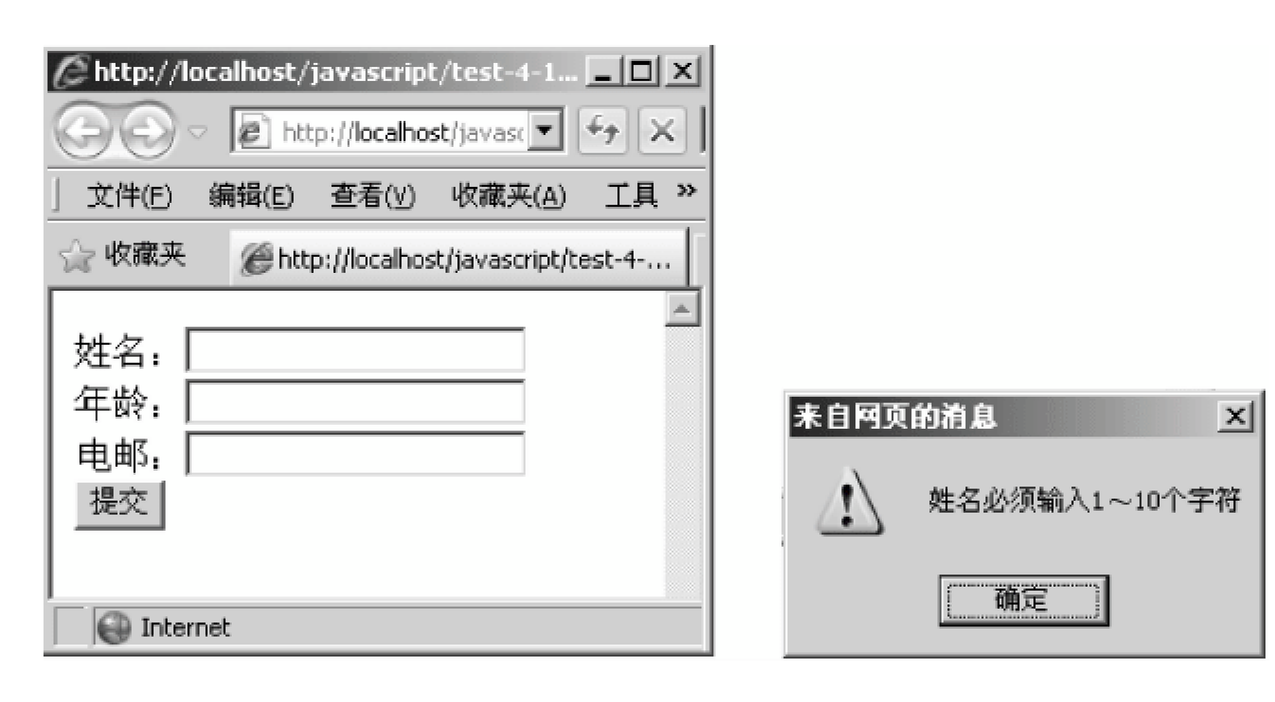


图 4-10　表单提交验证

4.4.2　向表格中动态添加行

在应用开发中，有时要输入数据，并动态添加到表格中，请看如下示例。

例 4-16　向表格中动态添加行(04-16.html)。

```
<html>
<head>
<meta http-equiv = "Content-Type" content = "text/html; charset = utf-8" />
<script type = "text/javascript">
function addrow()
{
  // 向表格中添加一个新行，并插入到第一行的位置
  var x = document.getElementById("myTable").insertRow(1)
```

```
    var y = x.insertCell(0)
    var z = x.insertCell(1)
    y.innerHTML = document.getElementById("tbxname").value;
    z.innerHTML = document.getElementById("tbxphone").value;
  }
</script>
</head>
<body>
姓名:<input id="tbxname" type="text" /><br/>
电话:<input id="tbxphone" type="text" /><br/>
<input type="button" value="添加" onclick="addrow()" /><br/>
<table id="myTable" border="1" width="160">
    <tr><td>姓名</td><td>电话</td></tr>
</table>
</body>
</html>
```

程序运行结果如图 4-11 所示，输入姓名和电话后，单击“添加”按钮，输入信息会自动添加到下面的表格中显示。

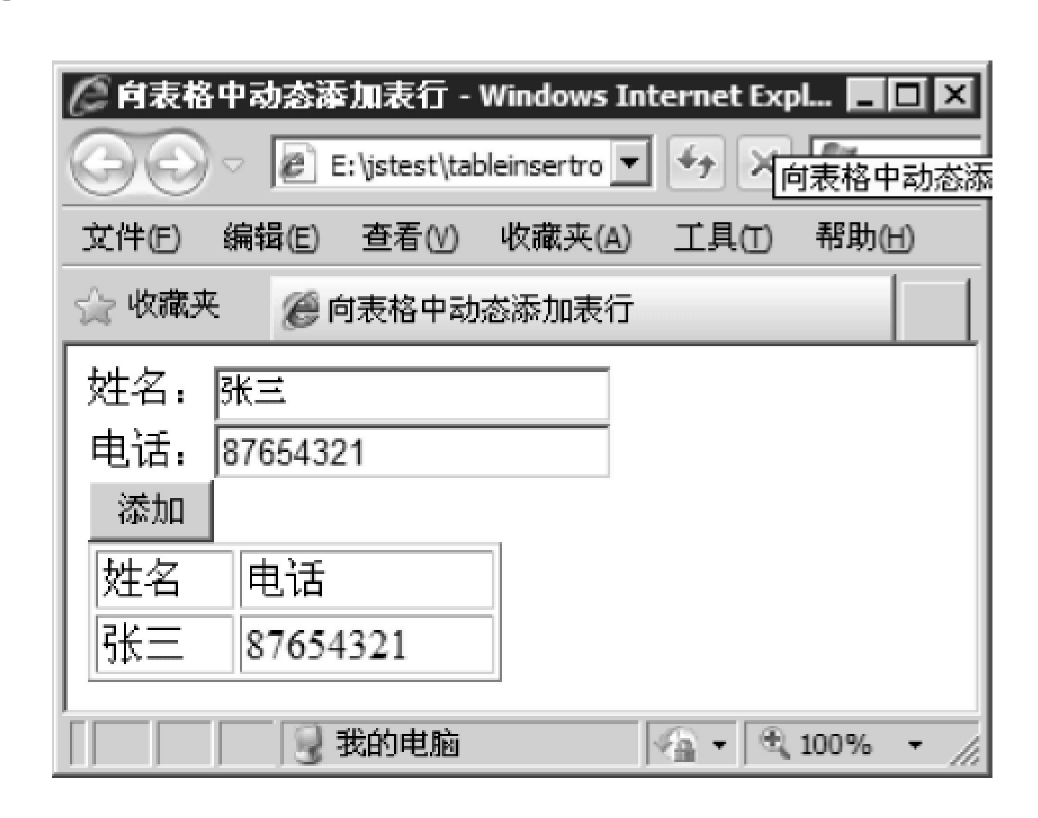


图 4-11　向表格中动态添加行

4.5　习题和上机练习

1. 选择题

(1) 写“Hello World”的正确 JavaScript 语法是(　　)。

A. document.write("Hello World")　　B. "Hello World"

C. response.write("Hello World")　　D. ("Hello World")

(2) 下列 JavaScript 的判断语句中(　　)是正确的。

A. if(i==0)　　B. if(i=0)

C. if i==0 then　　D. if i=0 then

(3) 下列 JavaScript 的循环语句中(　　)是正确的。

A. if(i<10;i++)　　B. for(i=0;i<10)

C. for i=1 to 10　　D. for(i=0;i<=10;i++)

(4) 下列选项中,(　　)不是网页中的事件。

A. onclick
B. onmouseover
C. onsubmit
D. onpressbutton

(5) 阅读以下 JavaScript 语句:

```
var a1 = 10;
var a2 = 20;
alert("a1 + a2 = " + a1 + a2)
```

将显示(　　)中的结果。

A. a1+a2=30
B. a1+a2=1020
C. a1+a2=a1+a2
D. 错误的表达式

(6) 某网页中有一个窗体对象,其名称是 mainForm。该窗体对象的第一个元素是按钮,其名称是 myButton,表述该按钮对象的方法是(　　)。

A. document.forms.myButton
B. document.mainForm.myButton
C. document.forms[0].element[0]
D. 以上都可以

(7) 在 HTML 页面上编写 JavaScript 代码时,应写在(　　)标签中间。

A. <javascript>和</javascript>
B. <script>和</script>
C. <head>和</head>
D. <body>和</body>

(8) 在 JavaScript 浏览器对象模型中,window 对象的(　　)属性用来指定浏览器状态栏中显示的临时消息。

A. status
B. screen
C. history
D. document

(9) 下列选项中关于浏览器对象的说法错误的是(　　)。

A. history 对象记录了用户在一个浏览器中已经访问过的 URL
B. location 对象相当于 IE 浏览器中的地址栏,包含关于当前 URL 地址的信息
C. location 对象是 history 对象的父对象
D. location 对象是 window 对象的子对象

(10) 在 HTML 页面中包含一个按钮控件 mybutton,如果要实现单击该按钮时调用已定义的 JavaScript 函数 compute,要编写的 HTML 代码是(　　)。

A. <input name="mybutton" type="button" onBlur="compute()"value="计算">
B. <input name="mybutton" type="button" onFocus="compute()"value="计算">
C. <input name="mybutton" type="button" onClick="function compute()" value="计算">
D. <input name="mybutton" type="button" onClick="compute()"value="计算">

(11) 分析下面的 JavaScript 代码段，输出结果是(　　)。

```
var mystring = "I am a student";
var a = mystring.substring(9,13);
document.write(a);
```

A. stud　　B. tuden　　C. uden　　D. udent

(12) 在 HTML 页面上，当按下键盘的任意一个键时都会触发 JavaScript 的(　　)事件。

A. onFocus　　B. onBlur

C. onSubmit　　D. onKeyDown

(13) 在表单(form1)中有一个文本框元素(fname)，用于输入电话号码，格式如 010—82666666，要求前 3 位是 010，紧接一个"—"，后面是 8 位数字。要求在提交表单时，根据上述条件验证该文本框中输入内容的有效性，下列语句中，(　　)能正确实现以上功能。

A.
```
var str= form1.fname.value;
if(str.substr(0,4)!="010—" || str.substr(4).length!=8 ||
    isNaN(parseFloat(str.substr(4))))
  alert("无效的电话号码!");
```

B.
```
var str= form1.fname.value;
if(str.substr(0,4)!="010—"  &&  str.substr(4).length!=8  &&
    isNaN(parseFloat(str.substr(4))))
  alert("无效的电话号码!");
```

C.
```
var str= form1.fname.value;
if(str.substr(0,3)!="010—" || str.substr(3).length!=8 ||
    isNaN(parseFloat(str.substr(3))))
  alert("无效的电话号码!");
```

D.
```
var str= form1.fname.value;
if(str.substr(0,4)!="010—" && str.substr(4).length!=8  &&
    ! isNaN(parseFloat(str.substr(4))))
  alert("无效的电话号码!");
```

2. 程序题

(1) 写出下列程序的运行结果。

```
function replaceStr(inStr, oldStr, newStr)
{
    var rep = inStr;
    while (rep.indexOf(oldStr) > -1)
    {
        rep = rep.replace(oldStr, newStr);
    }
    return rep;
}
alert(replaceStr("how do you do","do","are"));
```

(2) 当单击 button 按钮时,出现什么结果。

```
<html>
    <head>
        <title>Untitled Document</title>
        <script language=JavaScript>
            function add()
            {
                var first=document.myForm.first.value ;
                var second=parseInt(document.myForm.second.value);
                var third= parseInt(document.myForm.third.value);
                alert(first+second+third);
            }
        </script>
    </head>
    <body>
        <form name="myForm">
            <input type=text name="first" value="40">
            <input type=text name="second" value="30">
            <input type=text name="third" value="70">
            <input type=button value="add" onclick=add()>
        </form>
    </body>
</html>
```

(3) 补充按钮事件的函数,确认用户是否退出当前页面,确认之后关闭窗口。

```
<html>
  <head>
  <script  type="text/javascript" >
  function closeWin()
  {
  //在此处添加代码

  }
  </script>
  </head>
  <body>
    <input type="button" value="关闭窗口" onclick="closeWin()"/>
  </body>
</html>
```

(4) 完成函数 showImg,要求能够动态根据下拉列表的选项变化,更新图片的显示。

```
<body>
<script  type="text/javascript" >
function?? showImg (oSel)
{
//在此处添加代码

}
</script>
```

```
<img id="pic" src="img1.jpg" width="200" height="200" />
<br />请选择栏目
<select  id="sel" onchange="showImg(this)">
  <option value="img1">音乐</option>
  <option value="img2">体育</option>
  <option value="img3">财经</option>
</select>
</body>
```

3. 简答题

(1) 在页面中引入 JavaScript 有哪几种方式?

(2) 简要说明 JavaScript 的异常处理代码结构,并说明每一部分的作用。

(3) 简述文档对象模型中常用的查找访问元素节点的方法。

(4) 查阅资料,了解常用的 JavaScript 框架有哪些。

第 5 章　ASP.NET 基础

ASP.NET 是在 Microsoft .NET 框架的基础上构建、可提供构建企业级 Web 应用程序所需服务的一个 Web 平台。本章将主要介绍 Microsoft .NET 框架，构建 ASP.NET 的运行和开发环境，以及如何编写 ASP.NET 应用程序。

5.1　Microsoft .NET 框架

.NET 是一种面向网络、支持各种用户终端的开发平台环境，其目标是搭建新一代的因特网计算平台，解决网站间的协同合作问题，从而最大限度的获取信息。在.NET 平台上，不同网站间通过相关协定联系在一起，形成自动交流、协同工作，提供全面的服务。

Microsoft .NET 框架(.NET Framework)是一个集成在 Windows 中的组件，支持生成和运行下一代应用程序与 XML Web Services。

.NET Framework 具有两个主要组件：公共语言运行时(Common Language Runtime，CLR)和.NET Framework 类库(.NET Framework Class Library)。

1. 公共语言运行时

CLR 是.NET Framework 的基础，为执行.NET 脚本语言编写的代码提供了一个运行环境。CLR 管理.NET 代码的执行，提供内存管理、线程管理、代码执行、安全验证、远程处理等服务，并保证应用和底层操作系统之间必要的分离。同时，CLR 使得开发人员可以调试和进行异常处理。要执行这些任务，需要遵循公共语言规范(Common Language Specification，CLS)。CLS 描述了运行库能够支持的数据类型的子集。

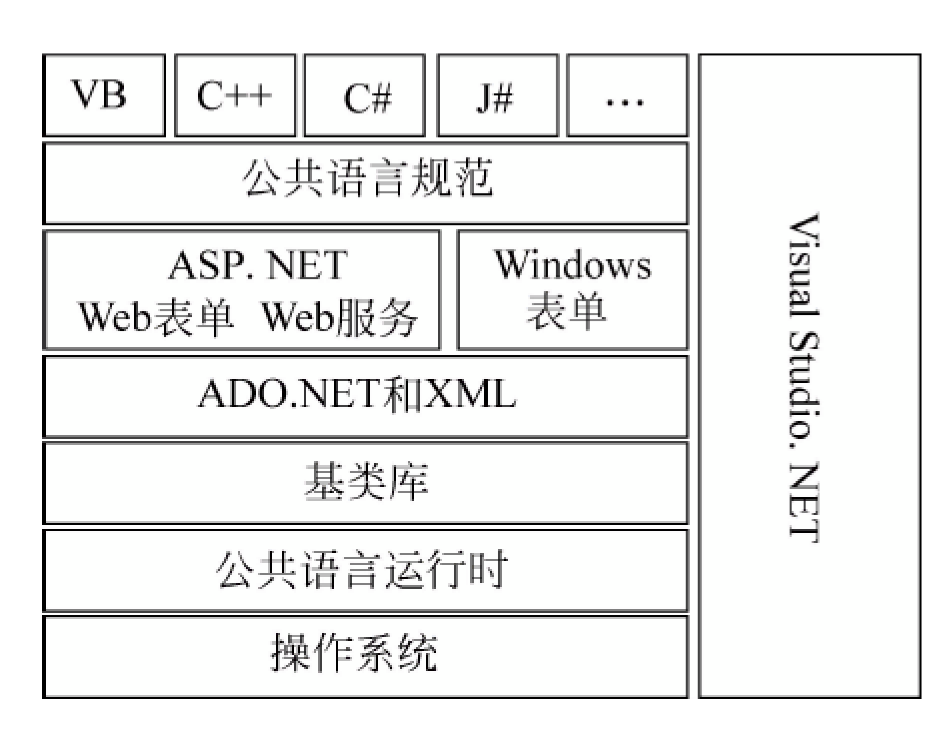


图 5-1　.NET Framework

在 CLR 监视之下运行的程序属于受控代码，也叫托管代码(Managed Codes)，不在 CLR 监视之下、直接在裸机上运行的应用或组件叫非托管代码(Unmanaged Codes)。

2. .NET Framework 类库

.NET Framework 类库是一个与 CLR 紧密集成、面向对象、可重用的类型集合，是生成.NET 应用程序、组件和控件的基础。.NET Framework 可以理解为一系列技术的集成，如图 5-1 所示。它包括.NET 脚本语言、CLS、.NET Framework 类库、CLR、Visual Studio .NET 集成开发环境等。

.NET Framework 是用于构建、开发以及运行 Web 应用程序和 Web Service 的公共环境，主要由三部分组成：编程语言、服务器端和客户端技术、开发环境。

1）编程语言(Programming Languages)

(1) C＃ (读作 C sharp)。

C＃是一种简洁、类型安全的面向对象的语言，是 Microsoft 公司专门为生成在.NET Framework 上运行的应用程序而设计的。C＃从 C 和 C++衍生而来，更像 Java。使用 C＃可以创建 XML Web Services、分布式组件、客户端/服务器应用程序、数据库应用程序等。

(2) Visual Basic .NET(VB .NET)。

Visual Basic.NET 是从 Visual Basic 语言演变而来，面向.NET Framework、能生成类型安全和面向对象应用程序的一种语言。VB .NET 是 Visual Studio .NET 的一部分，是一套完整的、可生成企业级 Web 应用程序的开发工具。

(3) J＃(读作 J sharp)。

J＃是一种供 Java 程序员构建在.NET Framework 上运行的应用程序和服务的语言。

2）服务器端和客户端技术(Server Technologies And Client Technologies)

(1) ASP .NET。

ASP.NET 是建立在.NET Framework 之上，利用 CLR 在服务器端为用户提供建立强大的企业级 Web 应用服务的编程框架。ASP.NET 主要包括 Web Form 和 Web Service 两种编程模型。前者提供建立功能强大的、基于 Form 的可编程 Web 页面。后者提供在异构网络环境下获取远程服务、连接远程设备、交互远程应用的编程界面。

(2) Windows Forms。

Windows Forms 是 .NET Framework 的智能客户端组件。利用 Visual Studio 之类的开发环境，使用 Windows Forms 可以创建应用程序和用户界面。Windows Forms 应用程序是基于 System. Windows. Forms 命名空间中的类、通过在窗体上放置控件并对用户操作(如单击)进行响应来构建的。

(3) Compact Framework。

Compact Framework 是为了在移动设备和嵌入式设备上运行而设计的，包含.NET Framework 中的类库的子集，同时还包含一些专有类。

3）开发环境(Development Environments)

(1) Visual Studio。

Visual Studio 是一个完整的集成开发环境(Integrated Development Environment, IDE)，用于生成 ASP.NET Web 应用程序、XML Web Services、桌面应用程序和移动应用程序。VB.NET、Visual C++.NET、Visual C＃.NET 和 Visual J＃.NET 全都使用相同的 IDE，该环境允许它们共享工具并有助于创建混合语言解决方案。

(2) Visual Web Developer。

Visual Web Developer 是一个功能齐备的开发环境，可用于创建 ASP.NET Web 应用程序。Visual Web Developer 提供网页设计、代码编辑、测试和调试，以及将 Web 应用程序部署到承载服务器等功能。

5.2 ASP.NET 概述

ASP.NET 是 Microsoft .NET 框架的重要组成部分，建立在公共语言运行库上，可用于在 Web 服务器上生成功能强大的 Web 应用程序，为 Web 站点创建动态的、交互的 HTML 页面。

5.2.1 ASP.NET 的发展历史

ASP.NET 的前身是 ASP(Active Server Pages)。ASP 是一种功能强大且易于学习的服务器端脚本编程环境。它是 Microsoft 公司的产品，在 NT Server、NT Workstation、Windows 2000 和 Windows XP 等系统中都附带这个脚本编程环境。ASP 的第一个版本是测试版。1996 年，ASP1.0 诞生，它作为 IIS 的附属产品免费发送。接下来，1998 年 Microsoft 又发布了 IIS 4.0 附带的 ASP 2.0 版本。ASP 1.0 和 ASP 2.0 主要区别是外部的组件需要实例化。2000 年，随着 Windows 2000 的成功发布，IIS 5.0 附带的 ASP 3.0 也开始流行。该版本不是简单地对 ASP 进行补充，最重要的是它把很多事情交给 COM 来做。

2000 年 6 月，Microsoft 公司总裁比尔·盖茨在一次名为"论坛 2000"的会议上发表演讲，描绘了.NET 的美景。

2001 年，Microsoft 公司在前面三个版本的基础上，推出了 ASP+，又叫 ASP.NET。这是全新的基于.NET 框架 的 ASP.NET 1.0 版本，使用 C# 作为默认语言。

2002 年，Microsoft 发布 ASP.NET 1.0 正式版，Visual Studio.NET 2002 也同步发行。

2003 年，推出 ASP.NET 1.1 版和 Visual Studio.NET 2003。

2004 年，发布.NET Framework 2.0 Beta1 和 Visual Studio 2005 Beta1。

2005 年，推出 ASP.NET 2.0 正式版，同时还有 Visual Studio 2005 和 SQL Server 2005 正式版。ASP.NET 2.0 的发布是.NET 技术走向成熟的标志。

2007 年，Microsoft 推出了 ASP.NET 3.0 版本，即.NET Framework 3.0。ASP.NET 3.0 就是在 ASP.NET 2.0 的基础上扩展了 WPF、WCF 和 WWF。

2007 年 11 月，推出 .NET Framework 3.5 和 Visual Studio 2008 正式版，加入了 LINQ(语言集成查询)并支持 ASP.NET AJAX，Web 开发技术又向前进了一大步。

2010 年 4 月，发布 Visual Studio 2010 正式版，同时推出.NET Framework 4.0、Visual Studio 2010 CTP，并且支持开发面向 Windows 7 的应用程序。除了 Microsoft SQL Server，它还支持 IBM DB2 和 Oracle 数据库。

5.2.2 ASP.NET 与 ASP 的区别

ASP.NET 又叫 ASP+，是在 ASP 的基础上发展起来的，但不是 ASP 的简单升级，而是 Microsoft 重新构筑的一个全新系统。ASP 存在着"先天"不足。例如，ASP 不是一个完全面向对象的系统，使用的脚本语言虽然简单、灵活，但属于弱类型语言，功能不强而且容易出错，内置对象少，ASP 代码的解释执行使得系统的执行效率比较低，等等。

ASP.NET 是建立在.NET 框架基础上的完全面向对象的系统。有了.NET 框架的支持,一些单靠应用程序设计很难解决的问题,都可以迎刃而解。.NET 框架平台给网站提供了全方位的支持,包括强大的类库、多方面服务的支持、多种语言进行开发、跨平台支持、充分的安全保障能力等。ASP.NET 程序采用 Visual Studio .NET 环境进行开发,支持 WYSIWYG(所见即所得)、拖放控件和自动部署等功能。

概括起来,ASP.NET 与 ASP 的区别主要体现在以下方面:

1. 执行效率

ASP 是解释执行的,ASP.NET 是编译执行的。ASP.NET 采用编译后运行的方式,执行效率大幅提高。它将程序在第一次运行时编译成 DLL 文件,以后直接执行 DLL 文件,这样速度就变得非常快。因此,ASP 在执行效率上大大低于实现同样功能的 ASP.NET。

2. 可重用性

ASP 将 HTML 代码和程序代码混在一起,代码的模块化和可重用性低;ASP.NET 将程序代码和 HTML 分开,实现了代码分离,程序结构清晰,代码的可重用程度高。ASP.NET 可以向目标服务器直接复制组件,当需要更新时,重新复制一个即可。

3. 代码量

ASP 所有功能都要编写代码,而 ASP.NET 提供了 Web 服务器控件,代码量大大降低。

ASP.NET 与现存的 ASP 保持语法兼容,实际上我们可以将 ASP 源文件扩展名.asp 改为.aspx,然后配置在支持 ASP.NET 运行的 IIS 服务器的 Web 目录下,即可获得 ASP.NET 运行时的优越性能。

5.2.3 ASP.NET 的工作原理

ASP.NET 的工作原理如图 5-2 所示。

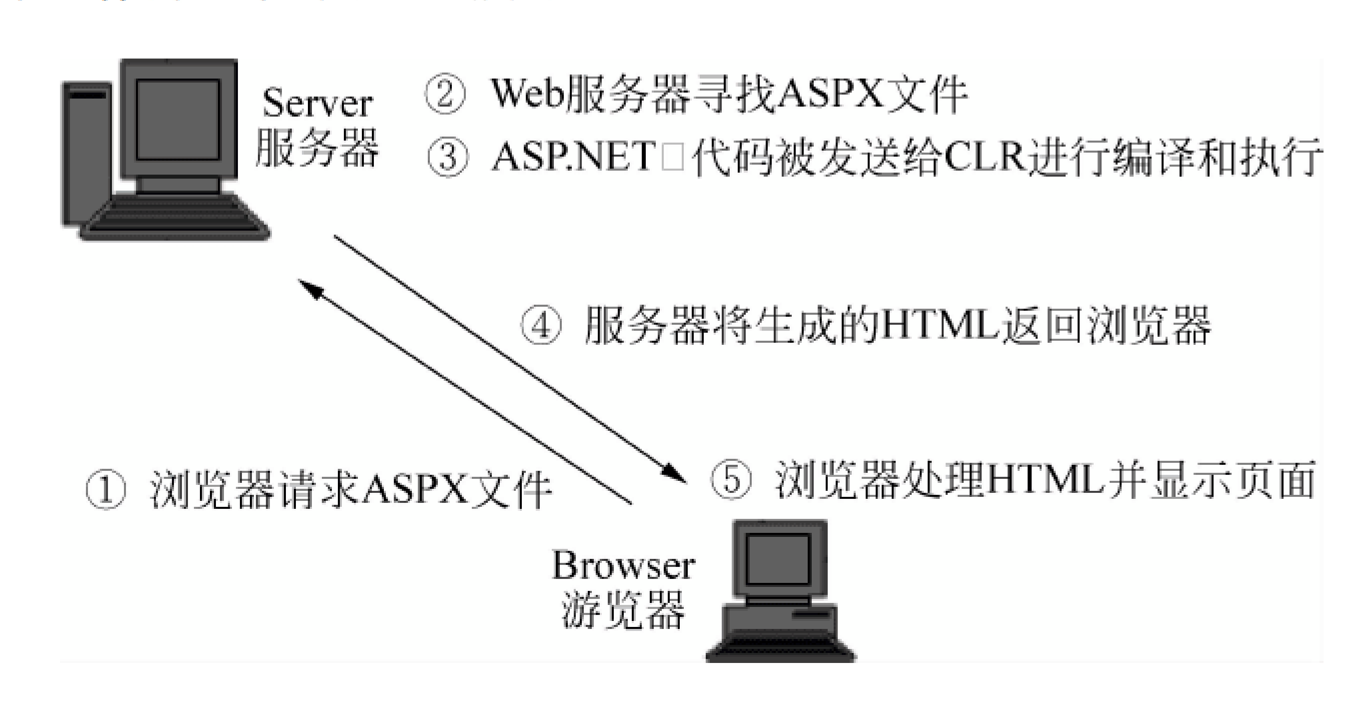


图 5-2 首次请求 ASP.NET 页面的处理过程

(1) Web 浏览器发送一个 HTTP 请求到 Web 服务器,要求访问一个 Web 网页。

(2) Web 服务器分析这个 HTTP 请求,定位所请求的 Web 网页的位置。

(3) 如果请求的网页是一个 HTML 文件,则服务器直接返回该文件。如果请求的是个 ASP.NET 文件,那么 IIS 就把该文件传送到 aspnet_isapi.dll 进行处理,后者把 ASP.NET 代码提交给 CLR。若是首次请求这个 ASP.NET 文件,就由 CLR 编译并执行,得到纯 HTML 结果;若是已经执行过该文件,那么就直接执行编译好的程序并得到 HTML 结果。

（4）最后把从（3）中得到的HTML文件传回浏览器作为HTTP响应。浏览器收到这个响应之后，就可以显示Web网页。

5.3 建立ASP.NET的运行和开发环境

要运行Web程序，必须首先建立一个Web服务器，然后从任一台Web浏览器访问该服务器上的Web程序。建立ASP.NET的运行环境，需要安装Web服务器IIS和.NET Framework。

要运行一个网站的程序，首先应在服务器上设置网站的根目录，并将网页文件放在该目录下，然后在IIS管理器建立一个指向网站根目录的虚拟目录，最后在浏览器地址栏输入相应的地址（通常包含虚拟目录）运行网站程序即可。

要编写和调试ASP.NET程序，通常选择Visual Studio集成开发环境。本书以当前主流配置的Visual Studio 2010和SQL Server 2008 R2为主要开发环境进行讲解。

5.3.1 安装和配置IIS服务器

IIS是ASP.NET唯一可以使用的Web服务器，目前常用的版本是IIS 6或IIS 7。下面以Windows XP为例，简述IIS 6的安装和配置步骤如下。

1. 安装IIS

安装步骤如下：

（1）选择“控制面板”中的“添加/删除程序”项，再选择“添加/删除Windows组件”。

（2）在“Windows组件向导”窗口中选中“Internet信息服务(IIS)”，单击“下一步”按钮，按照提示插入Windows安装光盘后，将自动安装IIS。

（3）安装结束后，在浏览器地址栏中输入http://127.0.0.1或http://localhost，如果出现IIS的信息页面，则表示IIS安装成功。此时，通常会在硬盘C上自动创建文件夹C:\Inetpub，这是IIS的默认目录。C:\Inetpub\wwwroot是默认的Web主页的地址。

2. 设置虚拟目录

假设一个ASP.NET文件hello.aspx存放在服务器上的某一目录D:\myWebSite中。

例5-1 一个简单的ASP.NET程序(hello.aspx)。

```
<%@page language="C#"%>
<html>
<body>
<font color=red>
<%
Response.Write("hello world.");
%>
</font>
</body>
</html>
```

下面介绍如何配置IIS，以及运行例5-1程序的方法。

① 选择“控制面板”中的“管理工具”项，再选择“Internet服务管理器”，进入“Internet

信息服务"主窗口。

② 展开"默认网站"项，选择"新建"→"虚拟目录"命令，启动"虚拟目录创建向导"程序，输入别名 mysite，并指向目录 D:\myWebSite，最后选择正确的访问权限。至此，建立了一个别名为 mysite 指向 D:\myWebSite 的虚拟目录。

最后，在浏览器中输入以下地址，就可以看到如图 5-3 所示的结果。

```
http://localhost/mysite/hello.aspx
```

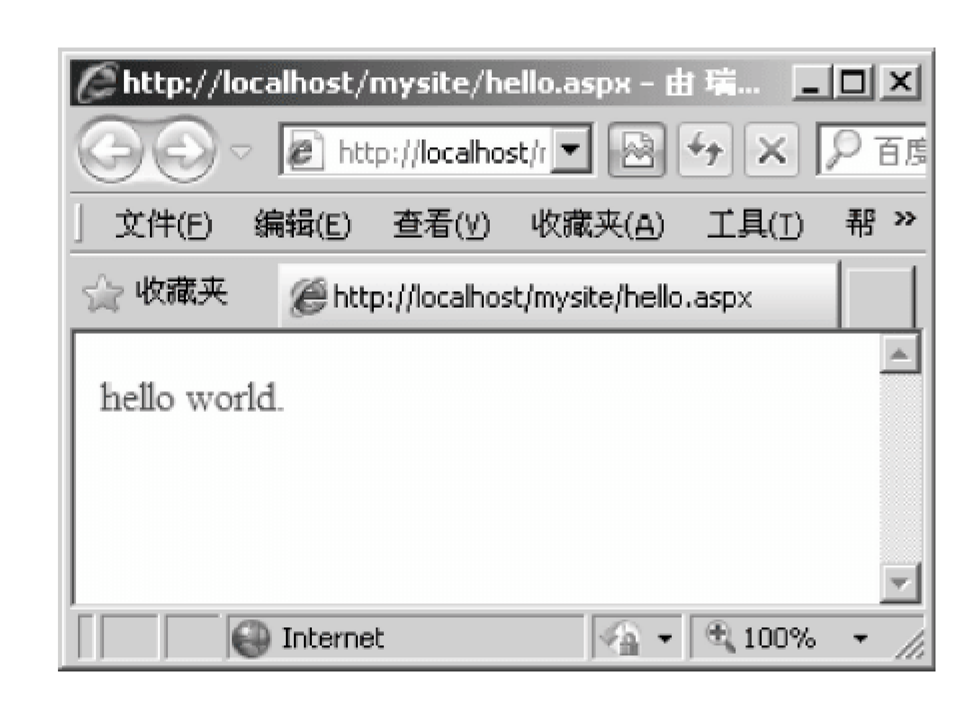


图 5-3　测试和运行 ASP.NET 程序

如果源文件放在二级目录下，如 D:\myWebSite\chapter5\hello.aspx，那么在浏览器中输入如下地址，也可以得到同样的运行结果。

```
http://localhost/mysite/ chapter5/hello.aspx
```

5.3.2　安装 Visual Studio 开发工具

要安装 Visual Studio 开发工具，必须首先安装 .NET Framework，然后再安装 Visual Studio 工具包。本书选择安装了 Microsoft Visual Studio 2010 开发工具包。

Microsoft Visual Studio 2010 安装程序通常自带 .NET Framework 4.0，后者也可以从 Microsoft 公司的官方网站 www.microsoft.com/downloads 下载。

Visual Studio 2010 目前有 5 个版本：专业版、高级版、旗舰版、测试专业版和速成版，如表 5-1 所示。

表 5-1　Visual Studio 2010 的版本

版　　本	说　　明
Professional（专业版）	面向个人的开发人员，供开发人员执行基本开发任务，提供集成开发环境、开发平台支持、测试工具等
Premium（高级版）	一个功能全面的、可扩展的工具集，可为个人或团队简化应用程序开发过程，支持交付可扩展的应用程序
Ultimate（旗舰版）	面向开发团队的、综合性的应用程序生命周期管理工具套件，通过利用高级协作功能、集成的测试和调试工具来确保整个团队从设计到部署的整个过程都能取得较高质量的结果

续表

版　　本	说　　明
Test Professional（测试专业版）	简化测试规划与人工测试执行的特殊版本，包含 TFS、ALM、MSDN 订阅、实验室管理、测试工具
Express（速成版）	一个免费的学习工具

基于普遍性的考虑，本书将安装 Visual Studio 2010 Ultimate 旗舰版，简称 VS 2010 Ultimate，可以通过下载一个 ISO 镜像文件进行安装。安装的初始界面如图 5-4 所示。



图 5-4　Visual Studio 2010 旗舰版的安装

Visual Studio 2010 Ultimate 包括以下工具，如图 5-5 所示。

- Visual Basic；
- Visual C++；
- Visual C#；
- Visual F#；
- Visual Web Developer。

本书选择安装 Visual Web Developer 和 Visual C#。

安装完成后，首次启动 VS 2010 Ultimate，选择默认的环境设置，然后进入如图 5-6 所示的起始页面。

单击“新建项目”项，接着选中 Visual C# 项，则进入如图 5-7 所示的页面。该页面分为左、中、右三个部分，在中间部分选择某个应用程序，并设定相应的文件名称，然后单击“确定”按钮进入相应的程序设计页面。



图 5-5　Visual Studio 2010 安装的选项页



图 5-6　Visual Studio 2010 的起始页



图 5-7 Visual Studio 2010 的"新建项目"对话框

通常,开发 ASP.NET 程序的第一步就是创建一个新网站。Visual Studio 2010 在"文件"菜单上比以前的版本做了很大改进,如图 5-8 所示。

选择"文件"→"新建网站"命令,打开一个"新建网站"对话框,进入如图 5-9 所示的界面。选择"ASP.NET 网站"项,在文本框中输入网站的文件夹位置,然后单击"确定"按钮,Visual Studio 将创建一个网站项目,并自动生成一个 Default.aspx 的网站默认主页。

从"网站"菜单中选择"添加新项",然后选择"Web 窗体",则添加一个 ASP.NET 页面。单击"添加"按钮,进入如图 5-10 所示的 Visual Studio 集成开发环境(IDE)主窗口。

在 IDE 主界面中,可分为 5 部分,图中用编号①~⑤进行标注。各部分的窗口界面都会按照用户的选择进行移动、悬靠和叠加,下面逐一说明。

(1) 区域①为"工具箱",悬靠在 IDE 的左边,用于选取控件。

(2) 区域②为"代码区",位于 IDE 的中间部分,可选择不同的视图设计源代码和页面代码。



图 5-8 Visual Studio 2010 的"文件"菜单

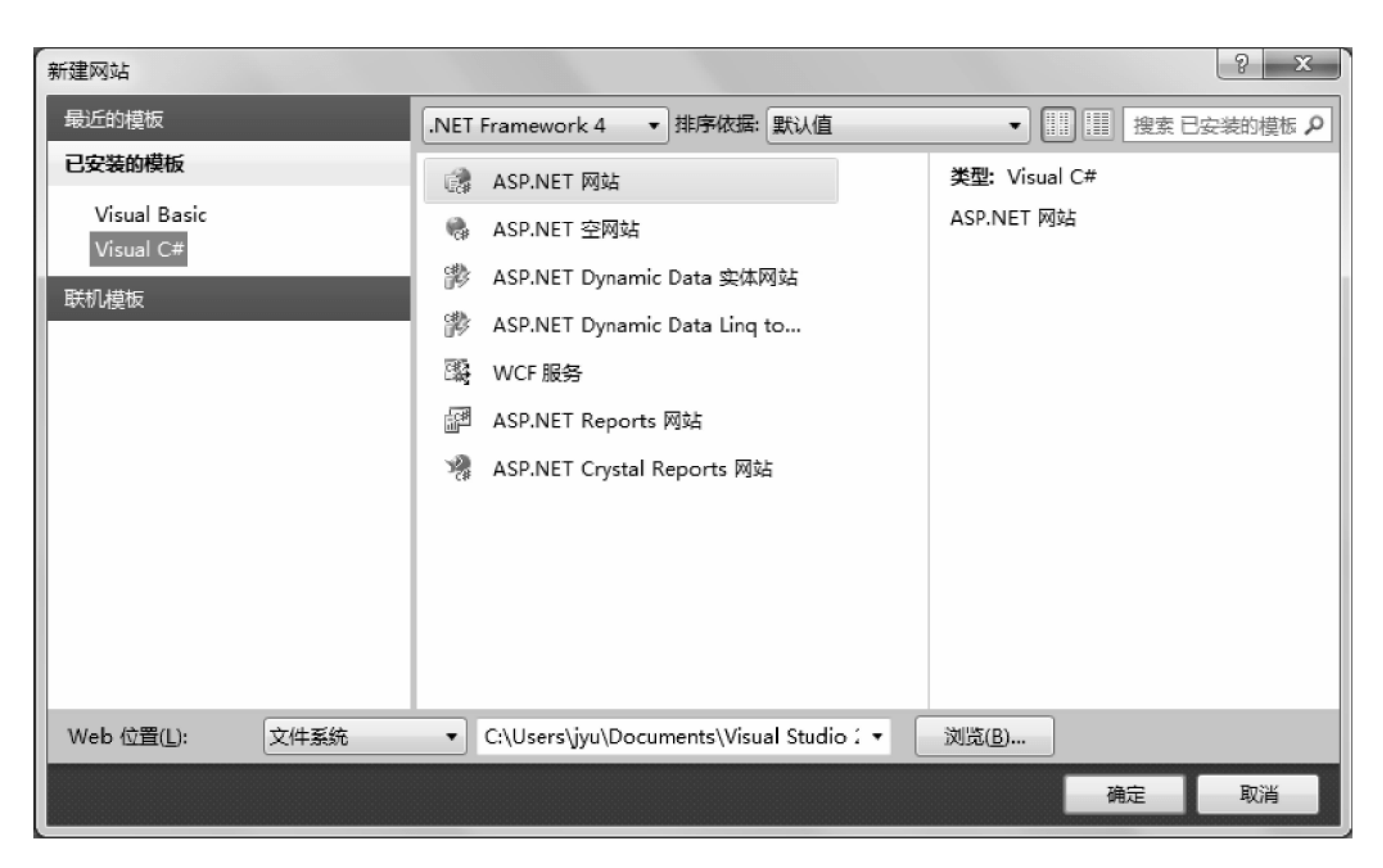


图 5-9 “新建网站”对话框

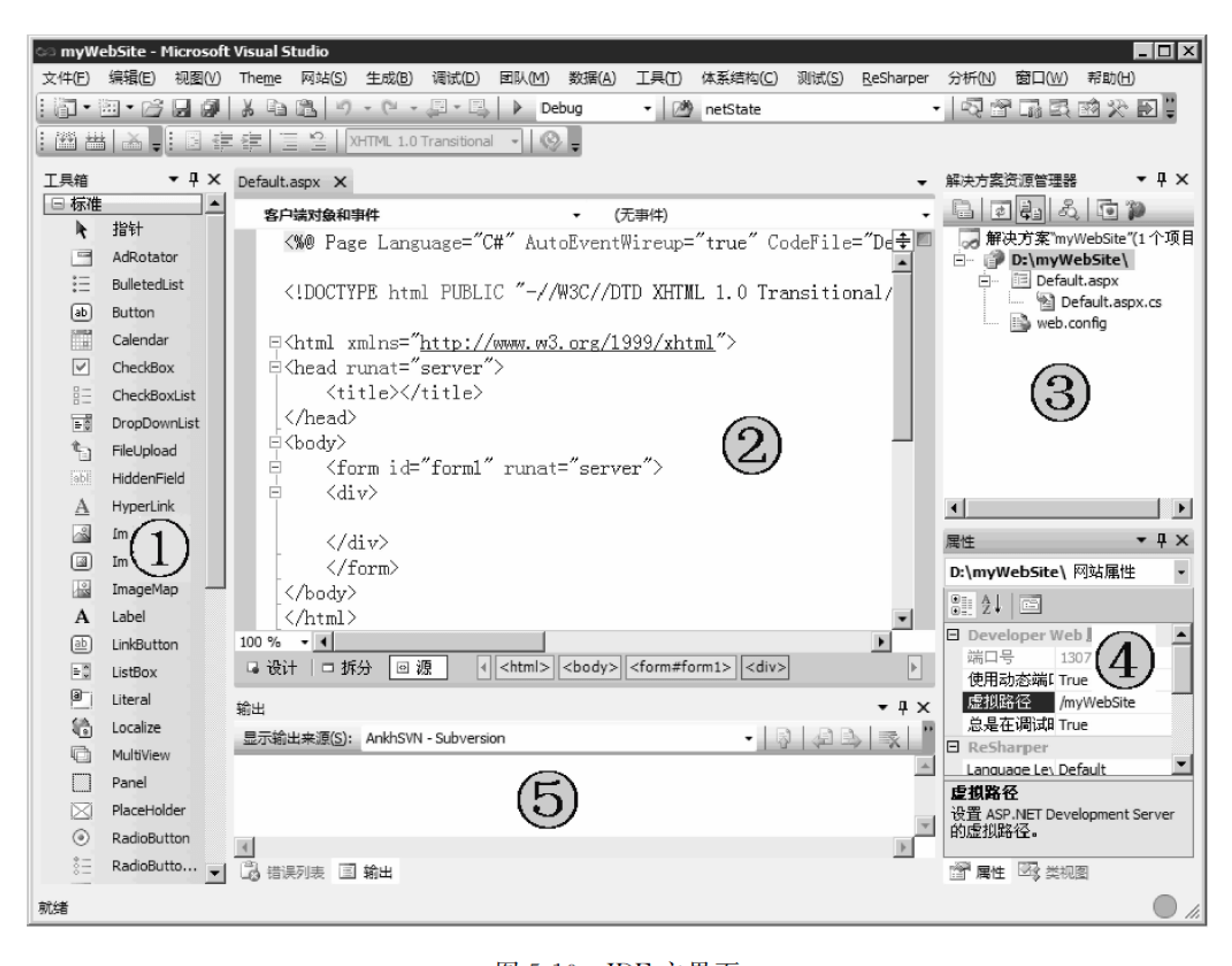


图 5-10 IDE 主界面

（3）区域③为“解决方案资源管理器”，一般悬靠在 IDE 的右上方，分不同选项卡进行显示。解决方案资源管理器用于显示项目中的文件。

（4）区域④为“属性窗口”，一般悬靠在 IDE 右下方，用于设置区域②里面选中控件的属性。

（5）区域⑤为“输出窗口”，悬靠在 IDE 下方，用于输出项目的编译信息。

在上述页面中，通过拖拽工具箱中不同的控件，以及编写 Web 页面的执行代码，就可以对 Web 页面进行自由的设计了。

5.3.3 SQL Server 数据库系统的安装

数据库服务器是安装了关系数据库的服务器，主要执行如数据存储、数据分析、查询检索等任务。微软公司目前主推 SQL Server 2008 R2 系统安装。其实，在安装 Visual Studio 2010 的过程中，已经自动安装了 SQL Server 2008 Express SP1 系统。也就是说，安装 VS 2010 时如果选择“完全安装”，那么它会安装 SQL Server 2008 的 Express SP1 版，这个版本只能通过 VS 2010 来访问使用，不能独立使用，但可以下载 Microsoft SQL Server 2008 Management Studio Express 免费的集成环境，用于访问、配置、管理和开发 SQL Server 所有组件，如图 5-11 所示。



图 5-11 SQL Server 2008 R2 的安装

此外，也可以选择单独安装一个完整的 SQL Server 2008。双击 exe 安装文件，进入如图 5-12 所示的页面，就可以开始安装过程了。



图 5-12　SQL Server 2008 安装界面

在实际安装过程中，需要特别注意的是数据库引擎的配置。

在数据库引擎配置中，最重要的就是“身份验证模式”的选择。

身份验证模式包括两种：Windows 身份验证模式和混合模式。

(1) Windows 身份验证模式。

该模式要求访问 SQL Server 的用户必须是经过 Windows 系统身份验证的用户。

(2) 混合模式(Windows 身份验证和 SQL Server 身份验证)。

既可以由 Windows 系统进行身份验证，也可以由 SQL Server 本身负责身份验证。当采用 SQL Server 验证方式登录数据库时，需要提供有效的账号和登录密码。

建议用户在安装时设定为“混合模式”，这样便于进行分布式访问。

5.4　开始编写第一个 ASP.NET 程序

5.4.1　Web 窗体代码模型

ASP.NET 页面可以用两种不同方式创建：单文件模式和后台代码模式。

1. 单文件模式

单文件模式类似 ASP 一样，一个页面对应一个文件。它将 HTML 代码和服务器端代码混合写在一个 *.aspx 的文件中，如例 5-1 中的 hello.aspx 文件。单文件模式具有方便修改、管理简单的优点，但当代码量非常大时，将大大降低代码的可读性，维护起来十分不便，因此，单文件模式适合于代码量较小的项目。

2. 后台代码模式

在 Visual Studio 环境中创建的 ASP.NET 页面通常为后台代码模式，即一个页面对应两个文件：*.aspx 和 *.aspx.cs 文件。前者只包含 HTML 和服务器控件标签；后者包含

服务器执行的后台代码。这种模式充分体现了"页面和代码分离"(Code Behind)的特点，非常有利于页面的布局和代码的维护，同时由于后台代码不暴露，也有利于代码的保密。代码的独立也使得它可在多个页面中重用。

在 Visual Web Developer 中，当选择"网站"→"添加新项"→"Web 窗体"时，出现如图 5-13 所示的页面。当取消选择右下方的"将代码放在单独的文件中"复选框时，文件命名为一个单独的.aspx 文件(如 Default1.aspx)。否则将自动建立两个文件，如 Default1.aspx 和 Default1.aspx.cs。



图 5-13 网站的"添加新项"窗口

5.4.2 ASP.NET 网页设计实例

下面通过一个简单的例子说明 ASP.NET 网页的设计过程。

1. 创建一个 ASP.NET 网站

启动 Visual Web Developer，选择"文件"→"新建网站"命令，在弹出的"新建网站"对话框中选择"ASP.NET 网站"，单击"确定"按钮，此时自动创建一个默认的 Default.aspx 网站主页和一个空的数据目录。

2. 创建 Web 窗体

设计一个新的 ASP.NET 页面，选择"网站"→"添加新项"→"Web 窗体"命令，取消选择"将代码放在单独的文件中"复选框，单击"添加"按钮，这时会自动创建 Default1.aspx 和 Default1.aspx.cs 两个文件。*.aspx 文件是对 Web 窗体的页面布局进行设计，*.aspx.cs 文件是对程序代码进行设计。

3. 设计 Web 窗体的页面(*.aspx)

图 5-14 所示为 Web 窗体的设计页面。单击"设计"标签，可以从工具箱中拖曳一个

TextBox 控件、一个 Button 控件、一个 Label 控件到设计窗口中，则看到当前页面的外观。单击“源”标签，可看到该页面的 XHTML 设计代码，即 Default.aspx 文件的内容。

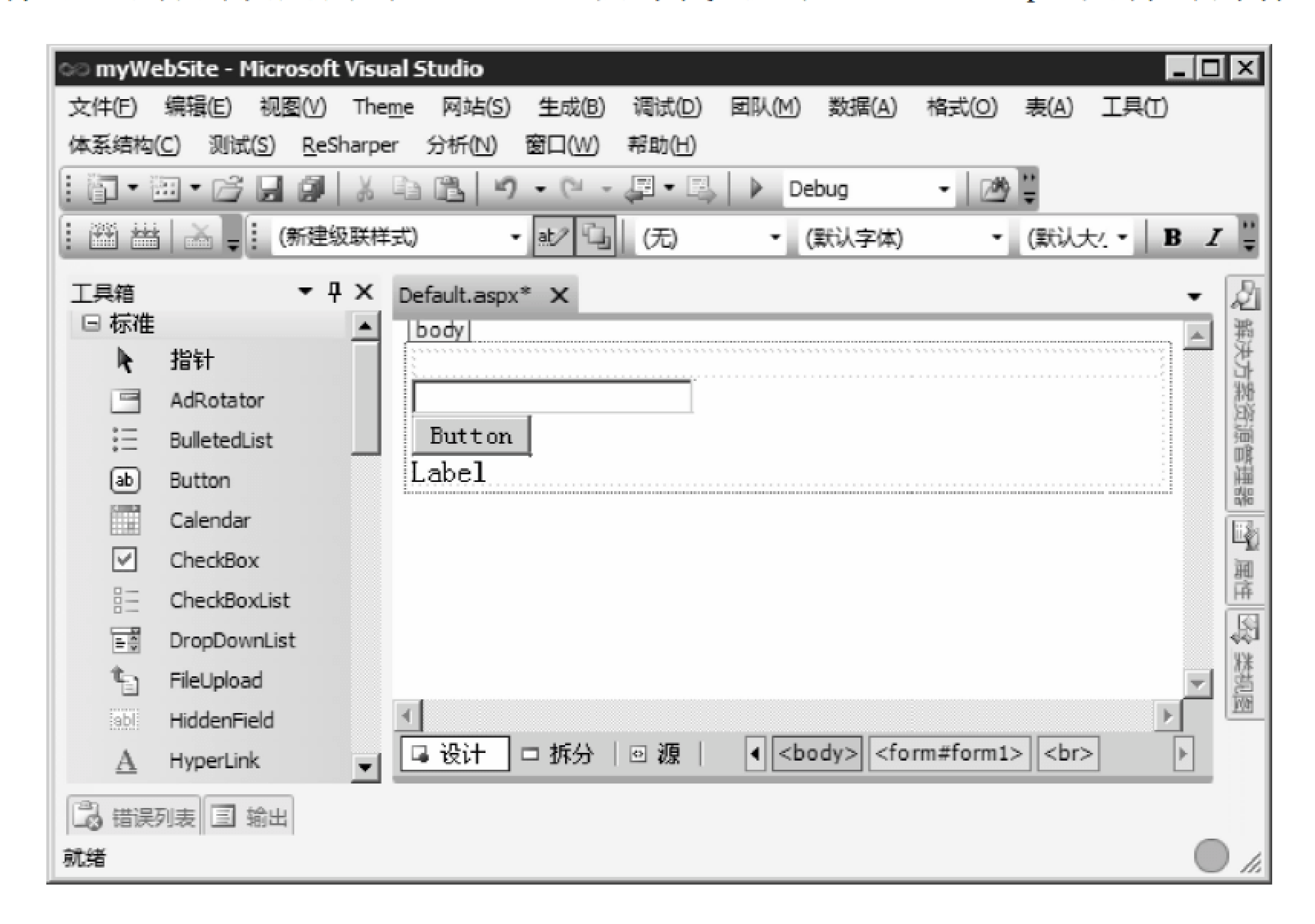


图 5-14 Web 页面的“设计”标签

4. 编写程序代码（*.aspx.cs）

在图 5-14 中，当双击 Button 控件，则可以打开 Default.aspx.cs 的设计页面，该页呈现了系统自动生成的默认代码内容。修改单击事件 Button1_Click，最后得到的代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Hello! " + TextBox1.Text;
    }
}
```

5. 运行程序

代码设计完成后，按 F5 键编译并运行应用程序。在文本框中输入内容，然后单击 OK 按钮，在 Label 上将显示程序的运行结果，如图 5-15 所示。

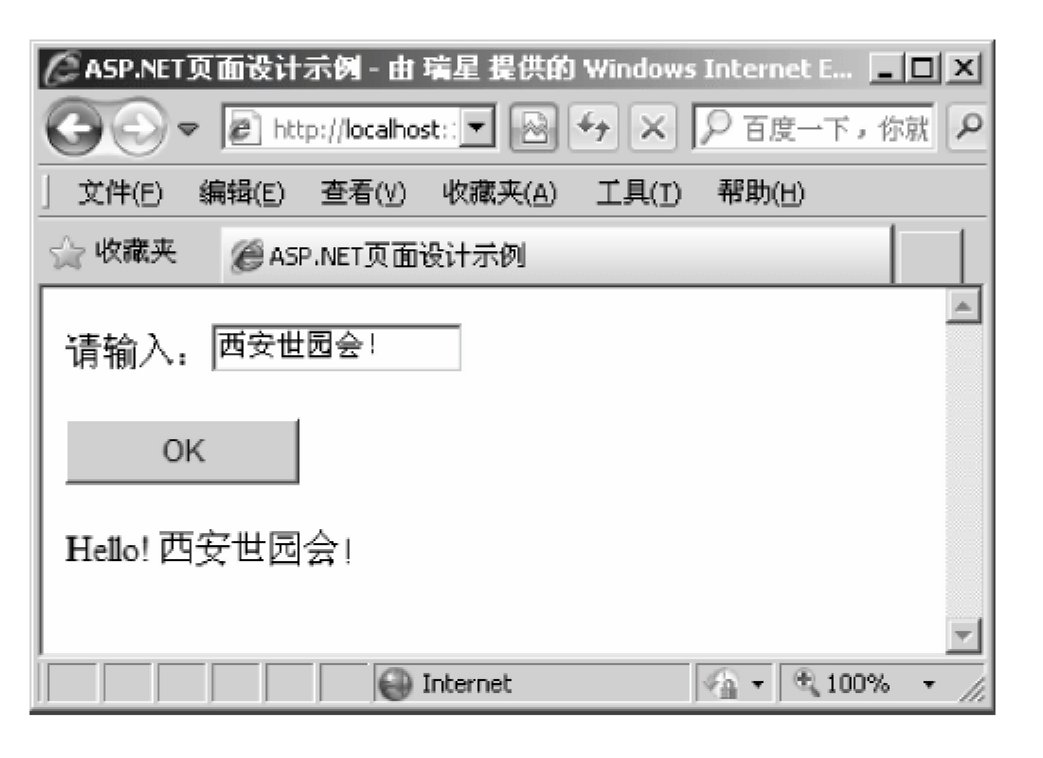


图 5-15　页面运行结果

5.5　习题与上机练习

1. 简答题

（1）简述.NET Framework 的作用和组成。

（2）什么是 Code Behind 技术？

（3）简述 ASP.NET 页面的处理过程，为什么第一次执行的时候，ASP.NET 程序执行很慢？

2. 上机练习

（1）使用.NET Web 窗体技术编写一个简单的个人主页。

（2）创建一个 Web 页面，用于显示当前服务器时间。

第6章 C# 语言基础

Microsoft .NET Framework 提供了至少4种以上的语言：Visual Basic.NET、Visual C#、Visual C++、Visual J#等。

C#是一种简单的、面向对象和类型安全的语言，是 Microsoft 公司专门为生成在.NET Framework 上运行的各种应用程序而设计的。C# 从C和C++衍生而来，它继承了C++最好的功能，比C++更简洁、高效。

Visual C# 是 Microsoft 对 C# 语言的实现。Visual C# 和 .NET Framework 的结合，使得程序设计人员可以创建 Windows 应用程序、XML Web Services、分布式组件、数据库应用程序等。Visual Studio 通过功能齐全的代码编辑器、编译器、项目模板、设计器、调试器等工具，实现了对 Visual C# 的支持。

6.1 创建一个简单的 C# 程序

先看一个简单的C# 程序，对C# 程序有一个初步的认识。

在菜单中选择“文件”→“新建项目”命令。在“新建项目”对话框中选择 Visual C#和“控制台应用程序”，然后填写程序的保存路径，如图6-1所示，单击“确定”按钮，系统将自动创建一个控制台程序的框架 Program.cs。

代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

代码的第1～第4行是引入命名空间。using 指令的作用就是引入命名空间，System 或 System.Collections.Generic 等就是名字空间。引入命名空间后，就可以直接使用它们的

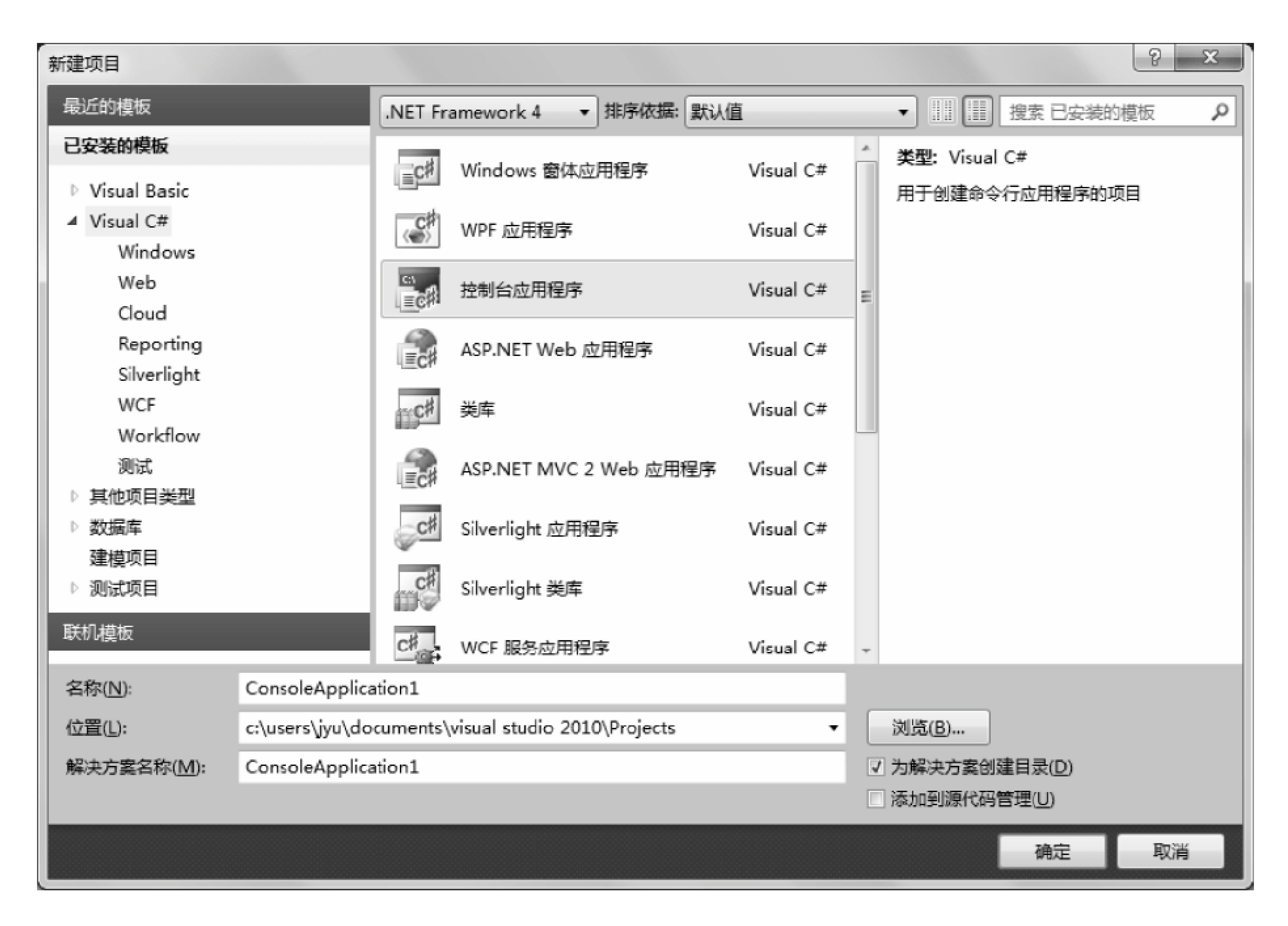


图 6-1　新建一个控制台应用程序

方法和属性了。建立一个控制台应用程序时，IDE 会自动引入常用的命名空间。

第 5 行 namespace ConsoleApplication1 是声明这个程序使用的命名空间。

第 6～第 13 行是用{}括起来的 C＃代码块。其中的 class Program 就是声明类名。一般类名和 cs 文件名相同，如果更改了 cs 文件名，IDE 会自动更新类名。类里面包含一个静态的 Main 方法，它是程序执行的起点和终点。

每一个 C＃程序都会包含一个 Main 方法，在 Main()中输入如下代码：

```
Console.WriteLine("Hello World ! ");
```

运行程序，则标准输出“Hello World !”字样。这里的 Console 就是 System 命名空间下的类，WriteLine 是 Console 类的方法。如果前面没有 using System 引入语句，则需要写成如下形式：

```
    System.Console.WriteLine("Hello World!");
```

通常，运行程序有两种方式：调试运行(按 F5 键)和不调试运行(按 Ctrl＋F5 键)。按 F5 键启动调试，程序运行结束后会立即关闭，如果发生异常，能定位异常出现的位置，还可设置断点让程序单步执行。按 Ctrl＋F5 键开始执行(不调试)，程序运行结束后提示“请按任意键继续…”，如图 6-2 所示，通常这种方式会忽略程序设置的断点。

需要附带说明的是，向控制台输出有如下几种常见方式：

```
Console.WriteLine();                          // 相当于换行
Console.WriteLine(要输出的字符串或变量);        // 输出一个值
Console.WriteLine("格式字符串",变量列表);       //格式化输出变量
```

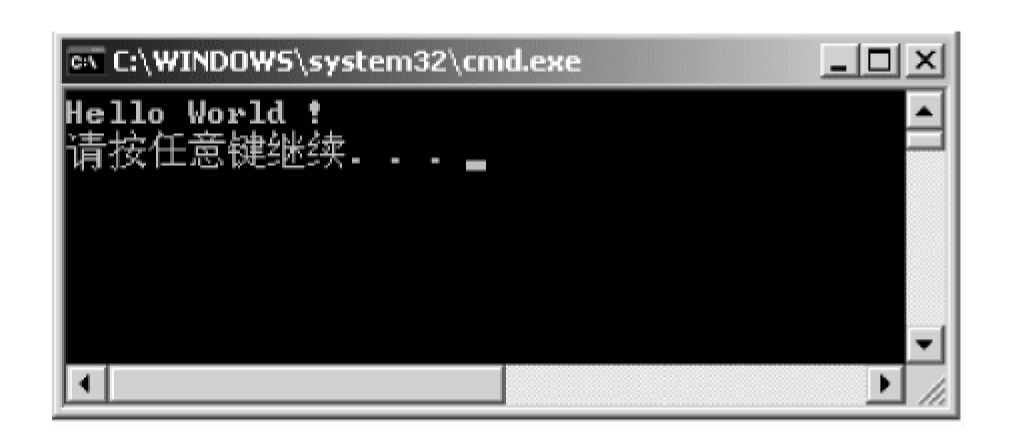


图 6-2　按 Ctrl+F5 组合键运行程序

例如：

```
Console.WriteLine ("Hello World!");                //输出一个字符串
string course = "C#";
Console.WriteLine(course);                         //输出一个变量
Console.WriteLine("我的课程名称是：{0}",course);     //格式化输出变量
```

6.2　C#基本语法

6.2.1　C#数据类型

C#中的数据类型分为两大类：值类型(Value Types)和引用类型(Reference Types)。值类型和引用类型的区别在于，值类型变量直接包含数据，而引用类型变量则存储对于对象的引用。

1. 值类型(Value Types)

值类型包含简单类型、结构类型和枚举类型。

1) 简单类型(Simple Type)

简单数据类型就是.NET 系统类型，表 6-1 所示为所有的简单数据类型。

表 6-1　C#的简单数据类型

类型	关键字	大小/精度	范围	.NET 类型	后缀
整型	byte	无符号 8 位整数	0～255	System. Byte	无
	sbyte	有符号 8 位整数	－128～127	System. SByte	无
	short	有符号 16 位整数	－32 768～32 767	System. Int16	无
	ushort	无符号 16 位整数	0～65 535	System. UInt16	无
	int	有符号 32 位整数	－2 147 483 648～2 147 483 647	System. Int32	无
	uint	无符号 32 位整数	0～4 294 967 295	System. UInt32	U 或 u
	long	有符号 64 位整数	－9 223 372 036 854 775 808～9 223 372 036 854 775 807	System. Int64	L 或 l
	ulong	无符号 64 位整数	0～0xffffffffffffffff	System. UInt64	UL

续表

类型	关键字	大小/精度	范围	.NET 类型	后缀
浮点型	float	32 位浮点值,7 位精度	$\pm1.5\times10^{-45}\sim\pm3.4\times10^{38}$	System.Single	F 或 f
	double	64 位浮点值,15～16 位精度	$\pm5.0\times10^{-324}\sim\pm1.8\times10^{308}$	System.Double	D 或 d
字符型	char	16 位 Unicode 字符		System.Char	无
布尔型	bool	8 位空间,1 位数据	true 或 false	System.Boolean	无
小数型	decimal	128 位数据类型,28～29 位精度	$\pm1.0\times10^{-28}\sim\pm7.9\times10^{28}$	System.Decimal	M 或 m

2）结构类型(Struct)

将所有相关的数据项(这些数据项的数据类型可能完全不同,称为域)组合在一起,形成一个新的数据结构,称为结构。

结构类型的声明格式如下：

```
struct 结构名
{
    public 数据类型 域名;
    …
};
```

下面的代码是一个典型的结构类型定义,定义了一个点的坐标。

```
struct Point
{
    public Double x , y , z ;
}
```

使用该结构类型时,可编写如下代码：

```
Point p ;
p.x = 100 ;
p.y = 200 ;
p.z = 300 ;
```

值得说明的是,结构类型不仅可以包含数据成员,还可以包含函数成员,这与类的定义十分类似。因此,结构类型的声明可以细化为如下形式：

```
struct 结构名
{
    public 数据类型 域名;
    …
    public void 方法名
    {
```

```
            //方法的实现
        }
};
```

下面是结构类型 student 的定义。所有与 student 关联的信息都作为一个整体进行存储和访问。

```
struct student
{
    public int stud_id;
    public string stud_name;
    public float stud_marks;
    public void show_details()
    {
        //显示学生详细信息
    }
}
```

关于类和结构的主要区别，总结如下：

- 结构是比类更简单的对象。与类一样，可以包含各种成员，也可以实现接口。
- 结构适合表示如点、矩形等简单的数据结构，这比使用类可以降低成本、效率更高。
- 最重要的差别在于，结构是“值类型”，而类是“引用类型”，结构不支持继承。
- 结构的实例化可以使用 new，也可以不使用 new(所有的域默认为 0，false，null 等)。而类的实例化必须使用 new。

3）枚举类型(Enumeration)

枚举类型是一组已命名的数值常量。使用这种方法，可以把变量的取值一一列出，变量只能在所列的范围内取值。

枚举类型的声明格式如下：

```
enum 枚举名
{
    //枚举元素列表
};
```

以下代码定义并使用了一个枚举类型 fruit。

```
enum fruit
{ Apple,Banana,Orange };   //值为 0、1、2
class EnumTest
{
  public static void Main( )
  {
  int choice;
  choice = (int)fruit.Apple;
  Console.Write("your choice:{0}", choice);
  }
}
```

枚举元素的默认基础类型为 int 型。在默认情况下，第一个枚举元素的数值为 0，后面每个枚举元素依次按 1 递增。在初始化过程中可重写默认值。

如果定义了下列枚举。

```
public enum WeekDays
{
    Monday,
    Tuesday,
    Wednesday = 20,
    Thursday,
    Friday = 5
}
```

那么，Monday 的值是 0，Tuesday 是 1，Wednesday 是 20，Thursday 是 3，Friday 是 5。

2. 引用类型(Reference Types)

和值类型相比，引用类型不存储实际数据，而是存储对实际数据的引用。引用类型包括对象、类、指代、接口、数组、字符串等。

1) 对象类型(Object)

在 C# 中，所有的类型都可以看成是对象，对象类型 Object 是一切类型的基类型。Object 类型对应的.NET 系统类型是 System. Object。

2) 类类型(Class)

类类型可以包含数据成员和函数成员。数据成员为常量、字段和事件。函数成员包括方法、属性、索引、操作符、构造函数和析构函数。

一个类可以派生多重接口。

3) 指代类型(Delegate)

指代类型可用于将方法用特定的签名封装。用户可以在一个指代实例中同时封装静态方法和实例方法。

指代的声明格式如下：

```
delegate 返回类型 代理名(参数列表)
```

代理的声明与方法的声明有些类似，这是因为代理就是为了进行方法的引用，但代理是一种类型。

例如：

```
Public delegate double MyDelegate(double x);
```

上述代码声明了一个代理类型，接着下面声明该代理类型的变量。

```
MyDelegate d;
```

对代理进行实例化的方法如下：

```
new 代理名(方法名);
```

其中,方法名可以是某个类的静态方法名,也可以是某个对象实例的方法名,但方法的返回值类型必须与代理类型中声明的一致。

例如:

```
MyDelegate d1 = new MyDelegate(System.Math.Sqrt);
MyDelegate d2 = new MyDelegate(obj.myMethod());
```

4) 接口类型(Interface)

一个接口定义一个只有抽象成员的引用类型。该类型不能实例化对象,但可以从它派生出类。

接口的声明格式如下:

```
interface 接口名
{
    //接口成员的定义
};
```

以下是一个接口定义例子。

```
Public interface MyInterf
{
        Void showface();
}
```

5) 数组类型(Array)

数组是一组类型相同的有序数据,可以存储整数对象、字符串对象或任何一种用户提出的对象。

声明数组时并不需要明确指定其大小,这样反而会出现编译错误。声明多维数组时每一维之间要用逗号隔开。

例如:

```
string [ ]   myarray = { "ab","aa","c","ddd"};          //一维数组
string [ , ]  twarray;                                  //二维数组
```

使用 new 关键字新建数组时,若指定了数组的大小,则大括号{ }中指定的元素个数必须相符,否则出错。若未指定大小,则根据{ }中元素个数自动分配大小。

例 6-1 使用一维数组和二维数组(06-01.aspx)。

```
<% @page language = "C#" %>
<script language = "C#" runat = "server">
void page_load(object serder, EventArgs e)
{
    int[] myArray1 = new int[5] {1,2,3,4,5};            //定义一个一维数组并赋值
    int[,] myArray2 = new int[2,3] {{1,2,3},{4,5,6}}; //定义一个二维数组并赋值
    labContent1.Text = myArray1[1].ToString();
    labContent2.Text = myArray2[1,2].ToString();
}
```

```
</script>
<html>
<body>
<asp:label runat = server id = labContent1/><br>
<asp:label runat = server id = labContent2/><br>
</body>
</html>
```

例 6-1 中，分别定义了一维数组和二维数组并赋以初值，然后将两个数组中的两个元素分别显示出来。最后显示的结果是 2 和 6。

6）字符串(String)类型

字符串类型就是 String 类型，是由一系列字符组成的。所有的字符串都是写在双引号中的。例如，“this is a book.”和“hello”都是字符串。

"A"和'A'有本质的不同，前者是 string 类型；后者是 char 类型。

3. 装箱与拆箱

由于在 C# 类型系统中，所有类型都是 Object 的子类型，这就为值类型和引用类型之间的转换提供了基础。

装箱就是将一个值类型隐式地转换为一个 Object 类型的过程。把一个值类型的值装箱，也就是创建一个 Object 实例，并将这个值复制该 Object 实例。

例如：

```
int i = 3;
Object o = i;
```

和装箱转换正好相反，拆箱就是将一个 Object 类型显式地转换成一个值类型。拆箱的过程分为两步：首先检查这个对象实例，看它是否为给定的值类型的装箱值；然后，把这个实例的值复制给值类型的变量。

例如：

```
int i = 0;
Object o = i;
int j = (int)o;
```

6.2.2 运算符和表达式

运算是对数据进行加工的过程，运算符就是描述各种不同运算的符号，参与运算的数据则称为操作数，操作数可以是常量、变量或者函数。

表达式由操作数和运算符组成。表达式的类型由运算符的类型决定，且每个表达式都产生唯一的值。在 C# 中，可以进行以下类型的运算，即算术运算、比较运算、(字符串)连接运算和逻辑运算等。

1. 算术运算符(Mathematical Operators)

算术运算符作用于整型或浮点型数据的运算。表 6-2 给出了算术运算符。

表 6-2　算术运算符(op：操作数)

运　算　符	说　　明	表　达　式
+	加法运算(若操作数是字符串,则为字符串连接符)	op1 + op2
-	减法运算	op1 - op2
*	乘法运算	op1 * op2
/	除法运算	op1 / op2
%	求余数	op1 % op2
++	将操作数加 1	op++,++op
--	将操作数减 1	op-- ,--op
~	将一个数按位取反	~op

2. 赋值操作符(Assignment Operator)

表 6-3 给出了赋值操作符。

表 6-3　赋值操作符(op：操作数)

运　算　符	说　　明	表　达　式
=	给变量赋值	op1 = op2
+=	运算结果 op1 = op1 + op2	op1 += op2
-=	运算结果 op1 = op1 - op2	op1 -= op2
*=	运算结果 op1 = op1 * op2	op1 *= op2
/=	运算结果 op1 = op1 / op2	op1 /= op2
%=	运算结果 op1 = op1 % op2	op1 %= op2

3. 比较运算符(Relational Operators)

比较运算符用于将表达式两边的值进行比较,其返回值为逻辑值 true 或 false。

C#有 6 个比较运算符,即等于(==)、不等于(!=)、小于(<)、大于(>)、小于等于(<=)、大于等于(>=)。

4. 逻辑运算符(Logical Operators)

逻辑运算符对布尔值 true 和 false 进行逻辑比较。共有 3 个逻辑运算符,如表 6-4 所示。逻辑运算的返回值是 true 或 false。

表 6-4　逻辑运算符

运算符	描　　述
&&	逻辑与(AND)
||	逻辑或(OR)
!	逻辑非(NOT)

5. 条件运算符

C#只有一个条件运算符,即三元操作符(? :),它是 if-else 语句的缩写。形式如下:

```
条件表达式?语句 1: 语句 2
```

如果条件表达式的值为真,则执行语句 1,否则执行语句 2。

下面的语句使用了条件运算符:

```
int result1,result2 ;
result1 = 10 > 1 ? 20 : 10 ;
result2 = 10 < 1 ? 20 : 10 ;
```

第一个条件表达式 10>1 为比较表达式，其值为 true，那么，三元运算符的返回值为第一个值 20，因此 result1 被置为 20。第二个条件表达式 10<1 的值为 false，那么三元运算符的返回值为第二个值 10，因此，result2 被置为 10。

6. 位运算符

位运算符对二进制位（0 或 1）进行比较和操作，共有 6 种运算符（如表 6-5 所示）。注意区分位运算与逻辑运算。“位与”运算用 1 个 & 符号，而“逻辑与”运算使用两个 & 符号。

表 6-5　位运算符

运　算　符	描　　述	运　算　符	描　　述
&	位与	～	位非
|	位或	<<	左移
^	位异或	>>	右移

对于位与运算（&）来说，比较两位二进制数，如果都是 1 的话，就返回 1；否则，返回 0，如表 6-6 所示。

表 6-6　位与运算的结果

位 1	位 2	位 1 & 位 2
0	0	0
0	1	0
1	0	0
1	1	1

类似地，对于其他的位运算符，其运算结果有这样的规律：

- 位或运算（|）比较两位，只要其中一位为 1，就返回 1；否则，就返回 0。
- 位异或运算（^）比较两位，只有在其中一位为 1 时，才返回 1；否则，返回 0。
- 如果位是 0 的话，位非运算（～）返回 1；否则，就返回 0。
- 左移运算（<<）把二进制的位向左移动指定的位数。移出左边的数被丢弃，而右边位补 0。
- 右移运算（>>）把二进制的位向右移动指定的位数。移出右边的数被丢弃，而左边位补 0。

下面的例子定义了两个叫 byte1 和 byte2 的变量：

```
byte1 = 0x9a;        //二进制数为 10011010，十进制数为 154
byte2 = 0xdb;        //二进制数为 11011011，十进制数为 219
```

注意：byte1 被设置为十六进制的 9a（十六进制数以 0x 开头），用二进制表示就是 10011010，用十进制表示就是 154。同样，byte2 被设置为十六进制的 db，也就是二进制的 11011011，十进制的 219。这些二进制和十进制数都显示在 byte1 和 byte2 赋值后的注释中。

设置字节变量 result 存储变量 byte1 和 byte2 的位运算结果如下。

```
result = (byte)(byte1 & byte2);
```

结果变量设为 10011010(十进制 154)。看一下二进制数:

```
    byte1  =  10011010   (154)
  & byte2  =  11011011   (219)
─────────────────────────────
    result  =  10011010  (154)
```

byte1 和 byte2 的每一位进行与运算。相应位的运算结果列在了下面。可以看到,最后的结果为二进制数 10011010,也就是十进制数 154。

例 6-2 位运算符(06-02.cs)。

```
class 06 - 02
{
public static void Main{}
    {
    byte byte1  =  0x9a;    //二进制数为 10011010,十进制数为 154
    byte byte2  =  0xdb;    //二进制数为 11011011,十进制数为 219
    byte result;
    System.Console.WriteLine("byte1  =  " + byte1);
    System.Console.WriteLine("byte2  =  " + byte2);
    result  =  (byte)(byte1 & byte2);        //与运算
    System.Console.WriteLine("byte1 & byte2  =  " + result);
    result  =  (byte)(byte1 | byte2);        //或运算
    System.Console.WriteLine("byte1 | byte2  =  " + result);
    result  =  (byte)(byte1 ^ byte2);        //bitwise exclusive OR
    System.Console.WriteLine("byte1 ^ byte2  =  " + result);
    result  =  (byte)~byte1;                 //逻辑非
    System.Console.WriteLine("~byte1  =  " + result);
    result  =  (byte)(byte1 << 1);           //左移
    System.Console.WriteLine("byte1 << 1  =  " + result);
    result  =  (byte)(byte1 >> 1);           //右移
    System.Console.WriteLine("byte1 >> 1  =  " + result);
    }
}
```

程序的输出如下:

```
byte1  =  154
byte1  =  219
byte1 & byte2  =  154
byte1 | byte2  =  219
byte1 ^ byte2  =  65
~byte1  =  101
byte1 << 1  =  52
byte1 >> 1  =  77
```

7. 运算符优先级

当一个表达式包含多个运算符时,将按照运算符的优先级顺序进行计算。表 6-7 列出

了按照优先级由高到低的顺序分组的 C＃运算符，每个组中的运算符具有相同的优先级，优先级为 1 的级别最高。

表 6-7　运算符的优先级

优先级	类别	运算符
1	基本	(x)　x.y　f(x)　a[x]　x++　x-- new　typeof　sizeof checked　unchecked
2	单目	+　-　!　~　++x　--x　(type)x
3	乘法与除法	*　/　%
4	加法与减法	+　-
5	移位	<<　>>
6	关系和类型检测	<　>　<=　>=　is　as
7	相等	==　!=
8	位与	&
9	位异或	^
10	位或	|
11	逻辑与	&&
12	逻辑或	||
13	三元	?:
14	赋值	=　*=　/=　%=　+=　-=　<<=　>>=　&= ^=　|=

在一个复杂的表达式中，具有高优先级的运算符总是先于低优先级的运算符进行计算。如果表达式包含多个相同优先级的运算符，则按照从左到右或从右到左的方向进行运算。例如，加运算符是从左到右进行计算，而赋值运算符和三元运算符是从右到左进行运算。

6.2.3　程序控制结构

在程序编写过程中，通常要根据条件的成立与否来改变代码的执行顺序，这就需要使用控制结构。C＃的程序控制语句包括分支语句、循环语句、跳转语句三类。

1. 分支语句

在分支语句中，可以先判断一个条件表达式的值，并根据判断的结果执行不同的程序代码块。C＃主要有两种分支语句，一个是实现双向分支的 if 语句；另一个是实现多分支的 switch 语句。

1) if 语句

if 语句的格式如下：

```
if (条件)
    { 语句段 1;  }
else
    { 语句段 2;  }
```

例 6-3 if 语句示例(06-03.aspx)。

```
<%@page language="C#"%>
<script language="C#"runat="server">
    void page_load(object serder, EventArgs e)
    {
       int intNowHour;
       intNowHour=DateTime.Now.Hour;
       if (intNowHour<12) labContent1.Text = "Good morning,Cindy!";
       if (intNowHour=12) labContent1.Text = "Good noon,Cindy!";
       if (intNowHour>12) labContent1.Text = "Good afternoon,Cindy!";
    }
</script>
<html>
<body>
    <asp:label runat=server id=labContent1 /><br>
</body>
</html>
```

上述代码定义了一个 int 型变量 intNowHour,用于保存当前时间的小时数。后面的三条 if 语句判定当前的时间是上午、中午还是下午,并分别给出问候信息。

与 JavaScript 相同,C#的 if 语句也支持嵌套,其嵌套形式如下:

```
if  条件表达式 1
    {语句序列 1}
else if  条件表达式 2
    {语句序列 2}
else if  条件表达式 3
    {语句序列 3}
    ...
else
    {语句序列 n+1}
```

说明: 如果 if 或 else 后有多条语句,必须使用大括号将其括起来。else 必须与 if 配对使用,也就是有几个 else 就必须有几个 if,而且每个 else 只与离它最近的一个尚未匹配的 if 配对。

2) switch 语句

与 JavaScript 中的 switch 语句相同,其语法格式如下:

```
switch (表达式)
{
    case 常量 1: 语句 1; break;
    case 常量 2: 语句 2; break;
  ...
    case 常量 n: 语句 n; break;
    default : 语句 n+1; break;
}
```

例 6-4 switch 语句示例(06-04. cs)。

```
using System;
class Sample
{
    public static void Main()
    {
       int myage = 10;
       string mystr;
       switch (myage) {
          case 10: mystr = "还是小孩!"; break;
          case 25: mystr = "可以结婚了!"; break;
          default: mystr = "不对吧!你到底多大!"; break;
       }
       Console.WriteLine("小子,你{0}",mystr);
    }
}
```

2. 循环语句

C# 中的循环语句主要有 while 语句、do-while 语句、for 语句、foreach 语句 4 种。

- while 语句：当条件为 True 时执行循环。
- do-while 语句：直到条件为 True 时执行循环。
- for 语句：指定循环次数,使用计数器重复运行语句。
- foreach 语句：对于集合中的每项或数组中的每个元素,重复执行。

1）while 循环

while 语句的格式如下：

```
while (<条件>)
{
    <循环体>
}
```

例 6-5 while 循环(06-05. cs)。

```
using System;
class Sample {
    public static void Main()
    {
       int sum = 0;
       int i = 1;
       while (i<= 100) {
           sum + = i;
           i++;
       }
       Console.WriteLine("从 0 到 100 的和是{0}",sum);
    }
}
```

2）do-while 循环

do-while 语句的格式为：

```
do
{
    <循环体>
}
while( <条件> );
```

例 6-6 do-while 循环(06-06. cs）。

```
using System;
class test
{
    public static void Main()
    {
       int sum = 0;     //初始值设置为 0
       int i = 1;       //加数初始值为 1
       do
        {
            sum + = i;
            i++;
        }
            while (i <= 100);
       Console.WriteLine("从 0～100 的和是{0}",sum);
    }
}
```

3）for 循环

for 循环表示的格式为：

```
for (循环变量赋初值; 循环条件; 循环变量增值)
{
    <执行语句>;
}
```

例 6-7 for 循环(06-07. cs)。

```
using System;
class test
    {
    public static void Main()
    {
      int sum = 0;
      for (int i = 1; i <= 100; i++)
      {
          sum + = i;
      }
      Console.WriteLine("从 0～100 的和是{0}\n",sum);
    }
}
```

4）foreach 循环

foreach 循环通过一个指定数据类型的变量，循环访问数组或集合中的元素。其基本语法格式如下。

```
foreach(type 变量名 in 集合)
{
    //循环体
}
```

例 6-8 foreach 循环(06-08. cs)。

```
using System;
class test_foreach
{
    public static void Main()
    {
        int[] myValues{2,3,4,1}
        foreach(int x in myValues)
        {
            Console.WriteLine("x= " + x);
        }
    }
}
```

该代码将在页面中显示：

```
x = 2
x = 3
x = 4
x = 1
```

3. 跳转语句

常见的跳转语句主要是 break 语句和 continue 语句，其他的有 return 和 goto 语句。

1）break 语句

break 语句跳出包含它的 switch、while、do、for 或 for-each 语句。

例 6-9 break 语句(06-09. cs)。

```
using System;
class test
{
    public static void Main()
    {
       int sum = 0;
       int i = 1;
       while (true)
       {
          sum + = i;
          i++;
          if (i>100)  break;   // 如果 i 大于 100,则退出循环
```

```
        }
        Console.WriteLine("从 0～100 的和是{0}",sum);
    }
}
```

上述代码中,如果没有 break 语句,while 循环将不会停止。在这个程序中,当 i 大于 100 时,则跳出循环,程序执行循环之后的下一个语句。

2）continue 语句

continue 语句用于结束本次循环,继续下一次循环,但是并不退出循环体。

例 6-10 continue 语句(06-10. cs)。

```
using System;
class test
{
    public static void Main()
    {
      for(int n = 100;n<= 200;n++)
      {
         if(n % 3 == 0)
         continue;
         Console.WriteLine("从 100～200 的不能被 3 整除的数是{0}",n);
       }
    }
}
```

上述代码中,当 n 是 3 的整数倍时,if 语句中的 continue 将被执行,这样立即跳出本次循环而开始了下一次循环。

6.3 类和对象

在程序中使用类和对象的好处,是可以模型化现实世界中的对象,即把对象的属性和行为封装在一个类中,这样可以减少解决复杂问题的难度。C♯是面向对象的程序设计语言,它使用类和结构来实现类型。典型的 C♯应用程序由程序员自定义的类和.NET Framework 提供的类组成。

6.3.1 类和对象的创建

类是相似的对象的一个组,类定义对象的属性和行为。类可以认为是一个模板,用来创建了对象。在 C♯中,属性保存在叫做“域”的变量中,行为则用“方法”来描述,两者都是类的成员。

类是一种数据结构,包含数据成员(变量、域和事件)和函数成员(方法、属性、构造函数和析构函数)。类的数据成员反映类的状态,而类的函数成员反映类的行为。

要创建一个类,在菜单中选择“文件”→“新建项目”命令,接着选择“Visual C♯”和“类库”,如图 6-3 所示。单击“确定”按钮,系统将自动创建一个类库的命名空间 ClassLibrary1,

并建立一个类文件名为 Class1.cs。

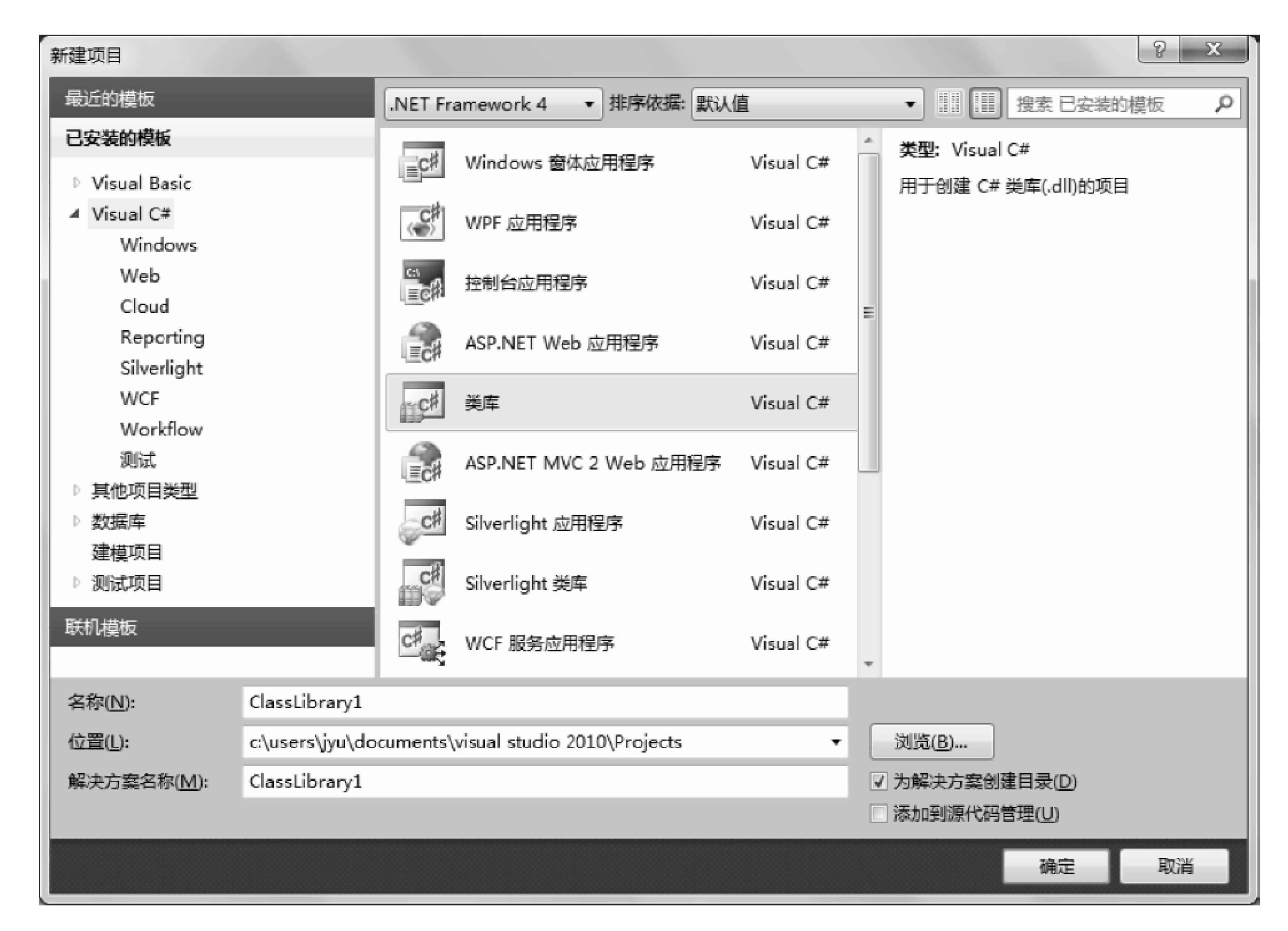


图 6-3　创建类

在新建的类文件 Class1.cs 中，默认的内容如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ClassLibrary1
{
    public class Class1
    {
    }
}
```

上述代码中，使用 class 关键字定义了一个类，类名为 Class1。此时可以在类体中定义类的数据成员和函数成员。

1. 类的声明

类的定义格式如下：

```
[访问权限符] class 类名 [: 基类名]
{
    <实例变量>
    <方法>
}
```

这里，访问权限符可以定义类的成员被其他类使用的权限。表 6-8 列出了常用的访问权限符，采用降序的方法，即 public 是访问权限最大的；private 是访问权限最小的。

表 6-8 访问权限符

访问权限符	说　明
public	完全公开，可以被所有的类所访问
internal	内部成员，只有本程序中的成员能够访问
protected	只有该类的派生类可访问，对其他类是隐藏的
private	只有该类的成员可以访问，任何别的类(包括派生类)都不能访问

下面的语句定义了一个叫 Car 的类。

```
public class Car
{
  //定义域(类的数据成员)
  public string model;
  …
  //定义方法(类的函数成员)
  public void Start()
  {
      System.Console.WriteLine(model + " started");
  }
  …
}
```

在类 Car 的定义中，每个域都有一个访问权限符，一般用 public 声明，表示对其存取无限制。方法也用 public 定义，表示对它的调用无限制。void 关键字表示不返回值。

2. 创建对象

1) 创建和访问对象

类是创建对象的模板，一旦创建了类，就可以创建那个类的对象。下面的语句创建了一个 Car 对象。

```
Car myCar;
myCar = new Car();
```

第 1 条语句声明了一个叫 myCar 的 Car 对象的引用，用来保存实际的 Car 对象的内存地址。第 2 条语句在计算机内存中实际创建 Car 对象。new 操作符为 Car 对象分配内存，Car 方法创建对象(也叫构造函数)。

还用一种简化的写法来创建对象。

```
Car myCar = new Car();
```

访问对象的域和方法使用点操作符(.)。

例如：

```
myCar.color = "red";        //给 color 域赋值
myCar. Start();             //调用方法 Start()
```

2）空值

当定义一个对象的引用时，其初始设置为 null(也可以当做“无引用”)，它并不是内存中的一个实际对象。例如，下面的语句声明了一个叫 myOtherCar 的对象的引用：

```
Car myOtherCar;
```

这里，myOtherCar 初始设置为 null，没有引用实际的对象(或者说，没有赋值)。此时，下面的行编译时会出错。

```
System.Console.WriteLine(myOtherCar.model);
```

也可以把 null 直接赋给一个对象引用。

例如：

```
myOtherCar = null;
```

它的意思是 myOtherCar 不再引用一个对象。程序不能使用 myOtherCar 对象，该对象将在资源回收的过程中被移出内存。

3）默认域值和初始值

当用类创建一个对象时，对象就将获得自己的类中声明的域拷贝。表 6-9 所示为各种类型的默认域值。

表 6-9　默认域值

类　型	默　认　值	类　型	默　认　值
所有数字类型	0	字符型	'\0'
布尔型	false	串	null

在声明一个类时，每个域都有一个系统默认值，可以通过赋初始值设置域的默认值。下面来看一个完整的类的定义和使用。

例 6-11　类的定义和使用(06-11.cs)。

```
using System;
public class Car
{
  public string model;          //定义域
  public string color;
  public int yearBuilt;
  public void Start()           //定义方法
  {
      System.Console.WriteLine(model + " started.");
  }
  public void Stop()
  {
      System.Console.WriteLine(model + " stopped.");
  }
}
public class Tester
```

```
{
 static void Main( )
 {
   Car myCar;                    //声明一个叫 myCar 的 Car 对象的引用
   myCar = new Car();            //创建 Car 对象，将它的内存地址保存到 myCar 中
   myCar.model = "Toyota";       //给 Car 对象的域赋值
   myCar.color = "red";
   myCar.yearBuilt = 2010;
   System.Console.WriteLine("myCar.model = " + myCar.model);
   System.Console.WriteLine("myCar.color = " + myCar.color);
   System.Console.WriteLine("myCar.yearBuilt = " + myCar.yearBuilt);
   myCar.Start();
   myCar.Stop();
   }
}
```

在上述代码中，设计了一个类 Car，同时实现了主类 Tester，这里通过 Main 函数定义了程序运行的入口。最后，程序的输出如下：

```
myCar.model = Toyota
myCar.color = red
myCar.yearBuilt = 2010
Toyota started.
Toyota stopped.
```

6.3.2 属性和方法

1. 类的数据成员和属性

在声明类的数据成员时，必须指明其访问级别，默认的访问级别是 private。若要使某些数据成员对外公开，则可由属性来实现。

换句话说，如果类的数据成员声明为 public，则用户可以在程序中直接、任意地访问该成员，导致类之间出现紧耦合。克服这一问题的方法是，定义私有的成员，再定义公有的属性，对其提供 get 和 set 访问。

属性的定义通过 get 和 set 关键字实现，get 用来定义读取属性时的操作，set 用来定义设置属性时的操作。

看下面的代码：

```
public class Car
{
  private string model;          //私有的数据成员
  public string color;
  public string Model            //公有的属性
  {
      get { return model;        //获取属性(提供读的权限)
  }
      set { return model = value ; //设置属性(提供写的权限)
```

```
        }
        }
        …
    }
```

如果一个属性同时具备了 get 和 set 操作，则该属性为读写性质的属性；如果只有 set 操作，则为只写属性；如果只有 get 操作，则为只读属性。

2. 类的方法

方法是执行一个任务的一组语句。在声明方法时，需要指定访问权限、返回值类型、方法名、使用的参数等。

1）方法的定义

声明方法的语句如下：

```
[访问权限符] [返回值类型] 方法名 (参数列表)
{ 方法体 }
```

方法通过 return 语句来返回值。如果方法没有返回值，则使用 void 关键字。

在例 6-11 定义的 Car 类中，只有两个无返回值的 Start()和 Stop()。下面的代码定义了一个带返回值的 Age 方法。

```
public class Car
{
    public int yearBuilt;
    public int Age(int currentYear)     // Age 方法计算并返回 Car 的已使用年限
    {
        int age = currentYear - yearBuilt;
        return age;
    }
}
```

2）方法的重载

通过方法的重载，可以在类中定义方法名相同而参数不同的方法。参数不同指的是参数的个数不同，或参数的类型不同。当一个重载方法被调用时，C＃会根据调用该方法的参数自动调用具体的方法来执行。

注意：*在 C＃ 中，方法的重载不关心返回值。也就是说，C＃ 不允许在一个类中存在两个方法名和参数列表相同、但返回值不同的方法。*

例 6-12　方法的重载(06-12.cs)。

```
using System;
class Overload{
    public void show()
    {
        Console.WriteLine( "nothing" );
    }
    public void show( int x )
```

```
        {
            Console.WriteLine( x );
        }
        public void show( string x, string y )
        {
            Console.WriteLine( x, y );
        }
        public static void Main( string[] args )
        {
            Overload myOverload = new Overload();
            myOverload.show();
            myOverload.show(3);
            myOverload.show("hello", "world");
        }
    }
```

上面代码中,第 1 个方法 show()没有参数,第 2 个方法 show()有 1 个 int 型参数,第 3 个方法 show()有 2 个 string 型参数。

6.3.3 构造函数和析构函数

构造函数在类被创建时自动执行(使用 new 语句时),析构函数在销毁类的时候被自动执行。

1. 构造函数(Constructor)

类的构造函数是这样的一种机制:用来在创建类的对象时赋予数据成员的值。构造函数是一种特殊的类成员函数,与类名相同,但不能有返回值。

构造函数用于执行类的实例的初始化。每个类都提供一个默认的构造函数。

使用构造函数请注意以下几个问题:

- 一个类的构造函数通常与类名相同;
- 构造函数不声明返回类型;
- 构造函数总是 public 类型的;
- 构造函数可以重载。

例 6-13 构造函数(06-13.cs)。

```
using System;
class Point
{
  public double x, y;
  public Point()
  {
      this.x = 0;
      this.y = 0;
  }
  public Point(double x, double y)
  {
      this.x = x;
```

```
        this.y = y;
    }
  }
  class Test
  {
    static void Main() {
    Point a = new Point();
    Point b = new Point(3, 4);     // 用构造函数初始化对象
  }
}
```

上述代码声明了一个类 Point，提供了两个重载的构造函数：一个是没有参数的 Point 构造函数；另一个是包含两个 double 参数的 Point 构造函数。如果类中没有提供这些构造函数，那么 C＃(更确切地说是 CLR)会自动创建一个默认的构造函数。注意，一旦类中提供了自定义的构造函数，如 Point() 和 Point(double x, double y)，则默认构造函数将不会提供。

2. 析构函数

析构函数是实现销毁一个类的实例的方法成员。析构函数不能有参数，不能加任何修饰符而且不能被调用。由于析构函数的目的与构造函数相反，就加前缀"～"以示区别。

虽然 C＃提供了一种新的内存管理机制——自动内存管理机制(Automatic Memory Management)，资源的释放是可以通过"垃圾回收器"自动完成的，一般不需要用户干预，但在有些特殊情况下还是需要用到析构函数的，如在 C＃中非托管资源的释放。

下面使用一段代码来表示析构函数是如何使用的：

```
public class ResourceHolder
{
  …
  ～ResourceHolder()
  {
    // 这里是清理非托管资源的用户代码段
  }
}
```

以下的例子综合使用了构造函数和析构函数。

例 6-14　构造函数和析构函数(06-14.cs)。

```
using System;
class Desk
{
  public Desk()                //构造函数和类名一样
  {
    Console.WriteLine("Constructing Desk");
    weight = 6;
    high = 3;
    width = 7;
    length = 10;
```

```
      Console.WriteLine("{0},{1},{2},{3}",weight,high,width,length);
    }
    ~Desk()                          //析构函数,前面加~
    {
      Console.WriteLine("Destructing Desk ");
    }
    protected int weight, high, width, length;
    public static void Main()  {
        Desk aa = new Desk();
        Console.WriteLine("back in main() ");
    }
};
```

6.3.4 继承和多态

1. 类的继承性(Inheritance)

继承的机制定义了类与类之间的父子关系。父类又称基类(Base Class),子类又称派生类(Derived Class),父类和子类之间形成了继承的层次体系。

在C#中,派生类从它的直接基类中继承成员:方法、域、属性、事件、索引指示器。除了构造函数和析构函数外,派生类隐式地继承了直接基类的所有成员,并在此基础上进行局部更改或扩充。

下面通过一个例子认识基类与派生类的继承关系。

例 6-15 类的继承(06-15.cs)。

```
using System;
class Vehicle                                  //定义汽车类
{
  int wheels;                                  //定义公有成员(车轮个数)
  protected float weight;                      //定义保护成员(重量)
  public Vehicle(){ ; }                        //构造函数
  public Vehicle(int w, float g)               //重载的构造函数
  {
    wheels  =  w;
    weight  =  g;
  }
  public void Speak()
  {
    Console.WriteLine("the w vehicle is speaking!");
  }
};

class Car:Vehicle                              //定义轿车类,即汽车类的派生类
{
    int passengers;                            //定义私有成员(乘客数)
    public Car (int w,float g,int p) :base(w,g)  //使用 base 保留字代表基类成员
    {
        wheels  =  w;
```

```
            weight = g;
            passengers = p;
        }
    }
    class Test
    {
        public static void Main( string[ ] args )
        {
            Car myCar = new Car();
            myCar.Speak ();
        }
    }
```

上述代码中，Vehicle 作为基类，体现了汽车实体具有的公共性质：汽车都有车轮和重量。Car 类继承了 Vehicle 的这些性质并且添加了自身的特性：搭载的乘客数。

C#中的继承符合下列规则：

- 继承是可传递的。如果 A 是基类，B 从 A 中派生，C 从 B 中派生，那么 C 不仅继承了 B 中声明的成员，同样也继承了 A 中的成员，Object 类作为所有类的基类。
- 派生类是对基类的扩展，可以添加新的成员，但不能除去已经继承的成员的定义。
- 构造函数和析构函数不能被继承。
- 派生类如果定义了与继承而来的成员同名的新成员，就可以覆盖已继承的成员。

2. 类的多态性(Polymorphism)

通过继承实现的不同对象调用相同的方法，表现出不同的行为，称为多态。

C#支持两种类型的多态性：编译时的多态，运行时的多态。

- 编译时的多态是通过重载来实现的，如方法重载和操作符重载；
- 运行时的多态是直到系统运行时，才根据实际情况决定实现何种操作。

编译时的多态性提供了运行速度快的特点，而运行时的多态性则带来了高度灵活和抽象的特点。

6.4 字　符　串

6.4.1 使用字符串

在程序中经常需要存储一系列的字符。通常使用 Unicode 格式的字符串来描述字符。Unicode 是为世界上绝大多数书写语言编码的标准，使用 16 位表示一个单词。

1. 创建字符串

下面的语句创建了一个叫 myString 的字符串：

```
String myString = "Hello World";
```

2. String 类的属性和方法

字符串实际上是 System.String 类的对象，可以在程序中使用其包含的属性和方法来操作字符串。表 6-10 给出了 String 类的属性和方法。

表 6-10 String 类的属性和方法

属性和方法	描　　述
Chars 属性	字符串索引器，获取当前 String 对象中位于指定字符位置的字符
Length 属性	字符串中的字符个数(只读)
Clone()	返回对此 String 实例的引用
Compare(String, String)	比较两个字符串
CompareOrdinal(String, String)	通过计算每个字符串中字符的数值来比较两个 String 对象
Compare(String, String)	比较两个指定的字符串，并返回一个整数
CompareTo(Object)	将此实例与指定的 Object 进行比较
Concat(String, String, String)	连接一个或多个字符串，构建一个新字符串
String1. Contains(String2)	判断字符串 String2 是否出现在字符串 String1 中，返回一个布尔值
Copy(String)	复制一个字符串 String
EndsWith(String)	确定字符串的结尾是否与指定字符串匹配
Equals(String, String)	判断两个字符串是否相等
Format(String, Object)	格式化字符串，即将字符串 String 的每项按 Object 的对应项替换
IndexOf(Char)	报告指定字符 Char 在字符串中第一次出现处的索引
Insert (int, String)	在字符串的指定起始位置(int)插入一个指定的 String 实例
Intern(String)	检索系统对指定 String 的引用
IsInterned()	返回一个对指定 String 的引用
Join(String, String[])	串联字符串数组的所有元素，在每个元素之间使用指定的分隔符
LastIndexOf(Char)	报告指定字符或字符串在此字符串中最后出现处的索引位置
Normalize()	返回一个新字符串，其二进制表示形式符合特定的 Unicode 范式
PadLeft(Int32)	返回一个新字符串，该字符串通过在字符左侧填充空格来达到指定的总长度，从而实现右对齐
PadRight(Int32)	返回一个新字符串，该字符串通过在此字符串中的字符右侧填充空格来达到指定的总长度，从而使这些字符左对齐
Remove(Int32)	从当前字符串中删除指定数量的字符，并返回新字符串
Replace(Char, Char)	在当前字符串中，用指定字符(或字符串)替换另一个字符(或字符串)，返回新字符串
Split(Char[])	返回的字符串数组包含此实例中的子字符串(由指定 Unicode 字符数组的元素分隔)
StartsWith(String)	确定字符串的开头是否与指定的字符串 String 匹配
Substring(Int32)	从指定的字符位置开始检索一个子字符串
ToCharArray()	从当前字符串复制字符到一个字符数组
ToLower()	字符串转换为小写形式
ToUpper()	字符串转换为大写形式
ToString()	将此实例的值转换为 String
Trim() 或 Trim(Char[])	从当前 String 对象的开始和结尾移除所有的空格或一组指定字符
TrimEnd()	功能同 Trim()类似，但仅仅移除尾部的所有空格或指定字符
TrimStart()	功能同 Trim()类似，但仅仅移除开头的所有空格或指定字符

下面简单介绍几个最常用的属性和方法。

1) 使用 Length 属性从字符串中读取单个字符

String 类有一个 Length 属性，表示字符串中的字符个数，返回一个 int 值。

通过指定一个字符在字符串中的位置(字符索引从0开始),从字符串中读取单个字符。例如,myString字符串被设置为"Hello World",则myString[0]为H。

下面的例子使用一个for循环来读取一个字符串的全部字符。

```
for (int count = 0; count < myString.Length; count ++ )
{
    Console.WriteLine("myString[" + count + "] = " + myString[count]);
}
```

2) 使用ToString方法把数据转换成字符串

ToString方法可以应用于任何.NET FrameWork所提供的数据类型,将之转换成字符串。一般来说,数据类型在转换时都是直接使用ToString方法,不带任何参数。但DateTime类型除外,它需要在ToString()中添加参数以选择输出日期的格式。此外,数字要想格式化输出,也要添加参数。

例如:

```
int age = 25;
string strAge = age.ToString();            // 整型转换成字符串
```

(1) 使用ToString方法格式化数字。

常用的参数及其含义如下:

```
C, c 表示货币,可指定小数点后位数。
F, f 表示定点计数法,指定小数位的位数。
X 表示十六进制。
```

例如:

```
double a = 17688.658
string str = a.ToString("C")       // 返回 ¥17688.658
str = a.ToString("C2")             // 返回 ¥17688.65
str = a.ToString("F2")             // 返回 ¥17688.65
```

(2) 使用ToString方法格式化日期和时间。

常用的参数及其含义如下:

```
D表示长日期,d表示短日期。
T表示长时间, t表示短时间。
F表示长日期和时间,f表示短日期和时间。
M,m表示月和日。
Y,y表示月和年。
```

例如:

```
DateTime dt = DateTime.Now
t = dt.ToString("D")            // 返回 Thursday,September 22,2011
t = dt.ToString("d")            // 返回 09/22/2011
t = dt.ToString("T")            // 返回 9:32:34 AM
```

```
t = dt.ToString("t")                    // 返回 9:32 AM
t = dt.ToString("f ")                   // 返回 Thursday,September 22,2011 9:26 PM
t = dt.ToString("yyyy 年 MM 月 dd 日")   // 返回 2011 年 09 月 22 日
```

3）使用 Compare 方法比较两个字符串

使用 Compare 方法的语法格式如下：

```
String.Compare( string1, string2 )
```

这里，string1 和 string2 是要比较的字符串，分别返回一个 int 值 1、0、－1 来指明第一个字符串大于、等于或小于第二个字符串。

例如：

```
int result1 = String.Compare("bbc","abc");   // Compare()返回 1
int result2 = String.Compare("abc","bbc");   // Compare()返回 - 1
```

如果要在比较中考虑字符串的大小写，可以使用如下语法：

```
String.Compare( string1, string2, ignoreCase )
```

这里，ignoreCase 是一个 bool 值，如果设置为 true（默认值），就无须考虑字符串的大小写；如果设置为 false，比较时就要考虑大小写。

例如：

```
int res1 = String.Compare("bbc", "BBC", true );  //忽略大小写,Compa re()返回 0
int res2 = String.Compare("abc", "BBC", false);  //考虑大小写,Compare()返回 - 1
```

4）连接字符串

（1）使用 Concat 方法连接字符串。

使用静态的 Concat 方法可以把字符串连接起来。该方法返回一个新的字符串，即把后面的字符串添加在前一个字符串的末尾。Concat()是可以重载的，最简单的语法如下：

```
String,Concat (string1, string2)
```

这里，string1 和 string2 是想连接在一起的字符串。Concat()中的参数也可以是三个字符串。

例如：

```
String myString4 = String.Concat("Friends,", "Romans");
        //字符串 "Friends,Romans"将存储在 myString4 中
String myString5 = String,Concat("Friends,", "Romans," , "and countrymen");
        //字符串 "Friends,Romans and countrymen"将存储在 myString5 中
```

（2）使用重载的加运算符来连接字符串。

也可以使用重载的加运算符（＋）来连接字符串。

例如：

```
String myString6 = "To be, " + "or not to be";
```

字符串"To be, or not to be"将存储在 myString6 中。

5）检查两个字符串是否相等

（1）使用 Equals 方法检查两个字符串是否相等。

使用 Equals 方法可以检查两个字符串是否相等，返回一个布尔值。它有两种格式，一个是在 String 类中调用 Equals()的静态版本；另一个是通过使用实际的字符串来进行比较的实例版本。其格式如下：

```
String.Equals(string1, string2)        //静态版本
String1.Equals(string2)                //实例版本
```

其中的 string1 和 string2 是要比较的两个字符串。

下面的例子中，mystring1 和 mystring2 是想要比较的两个字符串。

```
bool boolResult = String.Equals("bbc" , "bbc"):
boolResult = mystring1.Equals(mystring2);
```

（2）使用重载的等运算符来检查两个字符串是否相等。

可以使用重载的等运算符(= =)来检查两个字符串是否相等。下面的例子中，因为 myString 和 myString2 内容不同，boolResult 设置为 false。

```
boolResult = myString = = myString2;
```

例 6-16 字符串使用实例(06-16.cs）。

```
namespace Programming_CSharp
{
    using System;
    public class StringTester
    {
        static void Main( )
        {

            // 定义 3 个字符串
            string s1 = "abcd";
            string s2 = "ABCD";
            string s3 = @"Liberty Associates, Inc.
            provides custom .NET development,
            on - site Training and Consulting";
            int result;    // 保存比较结果

            // 比较两个字符串,区分大小写
            result = string.Compare(s1, s2);
            Console.WriteLine( "compare s1: {0}, s2: {1}, result: {2}\n", s1, s2, result);

            // 重载 compare 方法, 取布尔值 ignore case 为 (true = ignore case)
```

```
            result = string.Compare(s1,s2, true);
            Console.WriteLine("compare 大小写不敏感\n");
            Console.WriteLine("s4: {0}, s2: {1}, result: {2}\n",s1, s2, result);

            // 字符串连接方法
            string s6 = string.Concat(s1,s2);
            Console.WriteLine("s6 concatenated from s1 and s2: {0}", s6);

            // 重载操作符 +
            string s7 = s1 + s2;
            Console.WriteLine("s7 concatenated from s1 + s2: {0}", s7);

            // 字符串 copy 方法
            string s8 = string.Copy(s7);
            Console.WriteLine("s8 copied from s7: {0}", s8);

            // 使用重载后的操作符
            string s9 = s8;
            Console.WriteLine("s9 = s8: {0}", s9);

            // 使用 3 种方法进行比较
            Console.WriteLine("\nDoes s9.Equals(s8)?: {0}",s9.Equals(s8));
            Console.WriteLine("Does Equals(s9,s8)?: {0}",string.Equals(s9,s8));
            Console.WriteLine("Does s9 == s8?: {0}", s9 == s8);

            // 两种有用的属性: 索引和长度
            Console.WriteLine("\nString s9 is {0} characters long. ",s9.Length);
            Console.WriteLine("The 5th character is {1}\n",s9.Length, s9[4]);

            // 返回子串的索引值
            Console.WriteLine("\nThe first occurrence of Training ");
            Console.WriteLine ("in s3 is {0}\n",s3.IndexOf("Training"));

            // 在"training"之前插入单词 excellent
            string s10 = s3.Insert(101,"excellent ");
            Console.WriteLine("s10: {0}\n",s10);
        }
    }
}
```

代码输出结果如下：

```
compare s1: abcd, s2: ABCD, result: -1
compare insensitive
s4: abcd, s2: ABCD, result: 0
s6 concatenated from s1 and s2: abcdABCD
s7 concatenated from s1 + s2: abcdABCD
s8 copied from s7: abcdABCD
s9 = s8: abcdABCD
```

```
Does s9.Equals(s8)?: True
Does Equals(s9,s8)?: True
Does s9 == s8?: True
String s9 is 8 characters long.
The 5th character is A
s3:Liberty Associates, Inc.
provides custom .NET development,
on - site Training and Consulting
The first occurrence of Training
in s3 is 101
s10: Liberty Associates, Inc.
provides custom .NET development,
on - site excellent Training and Consulting
```

6.4.2 创建动态字符串

使用 System. Text. StringBuilder 类可以创建动态字符串。同 String 对象的一般字符串不同,动态字符串的字符可以被直接修改。String 对象是不可改变的,修改的总是字符串的拷贝。每次使用 System. String 类中的方法时,都要在内存中新建一个 String 对象,这就需要为新对象分配空间,增加了系统开销。如果要修改字符串而不创建新的对象,则可以使用 System. Text. StringBuilder 类提升性能。

因此,当进行频繁的字符串操作或操作很长的字符串时,使用 StringBuilder 类就比 String 类在效率上高很多。

1. 创建 StringBuilder 对象

下面的语句创建了一个叫 myStringBuilder 的 StringBuilder 对象。

```
StringBuilder myStringBuilder = new StringBuilder();
```

默认情况下,StringBuilder 对象初始可存储最多 16 个字符,但随着加入对象,其容量将自动增加。可以通过构建函数传递一个 int 参数来指定 StringBuilder 对象的初始容量;或传递两个 int 参数,其中第 2 个参数指定 StringBuilder 对象的最大容量。

例如:

```
int capacity = 50;
StringBuilder myStringBuilder2 = new StringBuilder(capacity);             //指定初始容量
int maxCapacity = 100;
StringBuilder myStringBuilder3 = new StringBuilder(capacity, maxCapacity);//最大容量
```

StringBuilder 对象的最大容量是 2 147 483 647(这也是 StringBuilder 对象的默认容量)。

可以通过传递一个字符串给构建函数设置 StringBuilder 对象的初始字符串:

```
string myString = "To be or not to be";
StringBuilder myStringBuilder4 = new StringBuilder(my String);
```

2. 使用 StringBuilder 对象的属性和方法

StringBuilder 类提供了许多属性和方法，如表 6-11 和表 6-12 所示。

表 6-11 StringBuilder 类的属性

属　性	类　型	描　述
Capacity	int	获取或设置 StringBuilder 对象中可以存储的最大字符数
Length	int	获取或设置 StringBuilder 对象中的字符数
MaxCapacity	int	获取 StringBuilder 对象的最大容量

表 6-12 StringBuilder 类的方法

方　法	返回类型	描　述
Append()	StringBuilder	在 StringBuilder 对象的结尾处添加字符串
AppendFornat()	StringBuilder	在 StringBuilder 对象的结尾处添加格式化字符串
EnsureCapacity()	int	确定 StringBuilder 对象的当前容量至少等于一个特定值，并返回一个 int 值，其中包括 StringBuilder 对象的当前容量
Equals()	bool	返回布尔值，指定 StringBuilder 对象是否等于一个特定对象
GetHashCode()	int	返回类型的 int 型哈希码
GetType()	Type	返回当前对象的类型
Insert()	StringBuilder	在 StringBuilder 对象的指定位置插入字符串
Remove()	StringBuilder	从 StringBuilder 对象的指定位置开始，删除特定数目的字符
Replace()	StringBuilder	在 StringBuilder 对象中，用字符串或字符代替出现的所有字符串或字符
Tostring()	String	将 StringBuilder 对象转换为一个字符串

可以看到，操作动态字符串的方法比操作一般字符串的方法少。

以下语句是错误的：

```
StringBuilder sb = "hello world!";           //不合法，不能这样初始化一个字符串
sb = "change the content";                   //不合法，不能直接把 String 转换成 StringBuilder
```

看下面的合法语句：

```
StringBuilder sb = new StringBuilder("Hello World! ");       //初始化字符串 sb
sb.Insert(6, "Beautiful ");          //将字符串"Beautiful "添加到当前指定位置
Console.WriteLine(sb);                                       //输出"Hello Beautiful World! "
sb.Remove(0, sb.Length);                                     //移除整个字符串
sb.Append("Test for string change!");                        //追加一个新字符串
int myInt = 25;
myStringBuilder.AppendFormat("...{0:C} ", myInt);
        //将一个设置为货币值格式的整数值放到 StringBuilder 的末尾
```

StringBuilder 类还有一个特性，它的 Length 属性不是 ReadOnly(只读)的，可以手动设置。在 String 类中，Length 属性是 ReadOnly 的。有这样的一组语句：

```
StringBuilder mysb = new StringBuilder("12345");  //初始化一个字符串 mysb
mysb.Length = 7;                                  //改变 mysb 的 Length 属性
```

```
Console.WriteLine("mysb(len = 7): {0}\n", mysb);     //输出 mysb 的内容为"12345"
mysb.Length = 3;
Console.WriteLine("mysb(len = 3): {0}\n", mysb);     //输出 mysb 的内容为"123"
```

6.5 集合编程

集合是 C# 中一个重要的数据组成形式,通过集合可以将数据存储于其中,并通过集合提供的特性,对数据进行索引、取值、排序等操作。System.Collections 命名空间包含这样一些集合类,如 ArrayList、哈希表、字典、堆栈、队列等,其对象创建以后还可以改变容量,同时提供了很多灵活的方法来操作和存取元素。

6.5.1 ArrayList

ArrayList 是 System.Collections 命名空间的一部分。ArrayList 可以理解为一种特殊的数组,与数组(Array)相似,都用于存储一组有序的数据元素。由于数组本身需要固定的长度,如果向其中增加元素则可能抛出异常,所以数组不够灵活。

ArrayList 集合可以动态地添加或删除所存储的元素。使用整数索引可以访问 ArrayList 集合中的元素,集合中的索引从 0 开始。

创建一个 ArrayList 集合对象时,不用定义其大小。ArrayList 有一个属性 Capacity,表示集合的容量,即能够存储的最多元素个数。ArrayList 集合的默认初始容量为 16,当添加第 17 个元素时,其容量自动翻倍到 32。你可以手工设置容量 Capacity,其值应该大于或等于元素个数, 如果设置的值小于元素个数,则程序将抛出一个异常 ArgumentOutOfRangeException。

创建一个 ArrayList 对象,可以使用如下方法:

```
ArrayList myArrayList = new ArrayList();
```

使用 Add 方法可以给 ArrayList 增加一个元素,并把新元素添加到 ArrayList 的末尾。下面的代码给 myArrayList 增加了两个字符串。

```
myArrayList.Add("Hello ")
myArrayList.Add("World ")
```

可以使用 Count 属性来获得存储在 ArrayList 中的元素个数。读取 ArrayList 中的元素时,可以在 for 循环中使用 Count 属性。

例如:

```
for (int i = 0; i<myArrayList.Count; i++)
{
    Console.WriteLine(myArrayList[counter]);
}
```

例 6-17 ArrayList 实例(06-17.cs)。

```
using System;
using System.Collections;
public class Employee                                      //定义一个类 Employee
{
    public Employee(int empID)  { this.empID = empID;  }  //构造函数
    public override string ToString( )  {  return empID.ToString( ); }
    public int EmpID
    {
        get { return empID; }
        set { empID = value; }
    }
    private int empID;
}

public class Tester
{
    static void Main( )
    {
        ArrayList empArray = new ArrayList( );
        ArrayList intArray = new ArrayList( );
        for (int i = 0;i<5;i++)   // 构造 intArray 和 empArray 集合中的元素
        {
            empArray.Add(new Employee(i+100));
            intArray.Add(i*5);
        }
        for (int i = 0;i<intArray.Count;i++)             // 打印 intArray 集合的全部内容
        {
            Console.Write("{0} ", intArray[i].ToString( ));
        }
        Console.WriteLine("\n");
        for (int i = 0;i<empArray.Count;i++)             //打印 empArray 集合的全部内容
        {
            Console.Write("{0} ", empArray[i].ToString( ));
        }
        Console.WriteLine("\n");
        Console.WriteLine("empArray.Capacity: {0}", empArray.Capacity);
    }
}
```

程序的输出结果如下：

```
0  5  10  15  20
100  101  102  103  104
empArray.Capacity:  16
```

6.5.2 哈希表

哈希表(Hash Table)表示一个关键码(Key)和值(Value)相关联的集合。也就是说，在哈希表中，每一个关键码都与一个值相对应，即 Key-Value 对。这就像字典一样，字典中的

单词就相当于关键码Key,对应的单词定义就是值Value。

建立一个哈希表,可以使用如下的方法:

```
Hashtable myHashtable = new Hashtable();
```

1. 添加元素

向哈希表中增加Key-Value对,可以使用Add方法。

```
myHashtable.Add("cn","China");
myHashtable.Add("hk","Hongkong");
myHashtable.Add("ca","Canada");
```

Add方法的第一个参数是关键码;第二个参数是值。但在添加元素时,如果使用了重复的关键码,则会给出一个异常ArgumentException。

2. 查找关键码对应的值

要想查找一个关键码对应的值,可使用索引来表示。例如,下面的代码在myHashtable表中查找一个关键码为hk对应的值。

```
String myCountry = (string) myHashtable["hk"];
```

查找的返回值是一个对象,在存储到myCountry变量之前被强制转换为字符串。

3. 获取关键码和值

要获取哈希表中的关键码和值,可以使用它的Keys属性和Values属性。

下面的语句中,利用foreach循环分别读取myHashtable的Keys属性和Values属性来显示哈希表中全部的关键码和值。

```
foreach (string mykey in myHashtable.Keys)        //显示全部的关键码
{
    Console.WriteLine("mykey = " + mykey);
}
foreach (string myValue in myHashtable.Values)   //显示全部的值
{
    Console.WriteLine("myValue = " + myValue);
}
```

哈希表有不少属性和方法,下面通过一个例子来了解它们的用法。

例6-18 哈希表的属性和方法(06-18.cs)。

```
using System;
using System.Collections;
class 06 - 18 - Hashtable
{
    pulic static void Main()
    {
        Hashtable myHashtable = new Hashtable();        //创建一个哈希表对象myHashtable
        myHashtable.Add("AL","Alabama");                 //添加key - value键值对
        myHashtable.Add("CA","California");
```

```
            myHashtable.Add("FL","Florida");
            myHashtable.Add("NY","New York");

            foreach (string myKey in myHashtable.Keys)        //显示哈希表的全部 Keys
            {
                Console.WriteLine("myKey  =  " + myKey);
            }
            foreach(string myValue in myHashtable.Values)     //显示哈希表的全部 Values
            {
                console.WriteLine("myValue  =  " + myValue);
            }
            if (myHashtable.ContainsKey("FL"))                //判断哈希表是否包含特定 Key
            {
                Console.WriteLine("myHashtable contains the key FL");
            }
            if (myHashtable.ContainsValue("Florida"))         //判断哈希表是否包含特定 Value
            {
                Console.WriteLine("myHashtable contains the value Florida");
            }

            Console.WriteLine("Removing FL from myHashtable");
            myHashtable.Remove("FL"):                      // 移除哈希表中的某个 key－value 键值对
            int count  =  myHashtable.Count;                  // 获得哈希表的元素个数

            Console.WriteLine("Copying keys to myKeys array");
            string[] myKeys  =  new string[count];
            myHashtable.Keys.CopyTo(myKeys, 0);   //将哈希表的 Keys 复制到数组 myKeys 中
            for (int counter = 0; counter < myKeys.Length; counter++)  //显示数组 myKeys 的内容
            {
                Console.WriteLine("myKeys[" + counter + "] = " + myKeys[counter]);
            }
        }
    }
```

程序的输出结果如下：

```
myKey = AL
myKey = CA
myKey = FL
myValue = Alabama
myValue = California
myValue =  Florida
myHashtable contains the key FL
myHashtable contains the value Florida
Removing FL from myHashtable
Copying keys to myKeys array
myKeys[0]  =  AL
myKeys[1] = CA
myKeys[2] = NY
```

6.5.3 队列

队列(Queues)是一个遵循"先进先出"(First In First Out)原则的集合。在一端输入数据称为加队(Enqueue);在另一端输出数据称为减队(Dequeue)。可见,队列中数据的插入和删除都必须在队列的头尾进行,而不能直接在中间的任意位置插入和删除数据。

在管理有限的资源时,队列是一个非常好的数据结构。例如,当需要在只有一个CPU的计算机系统中运行多个任务时,由于计算机一次只能处理一个任务,其他的任务就被放在一个专门的队列中排队等候。另外,打印机缓冲池中的等待作业队列,也是使用队列的例子。

创建一个Queue对象,可以使用如下方法:

```
Queue myQueue = new Queue();
```

使用Enqueue方法可以添加元素到队列尾。

例如:

```
myQueue.Enqueue("This");
myQueue.Enqueue("is");
myQueue.Enqueue("a");
myQueue.Enqueue("test");
```

这些元素在队列myQueue中的顺序为This、is、a、test。

使用Dequeue方法可以删除队列头的元素。该方法返回这个元素,然后从队列中删除它。以下代码将显示"This",它也将从myQueue中删除。

例如:

```
Console.WriteLine(myQueue.Dequeue());
```

读取队列中最前面的元素,可以使用Peek方法。该方法也返回这个元素,但并不从队列中删除它。下面的代码将会显示"is",该元素在队列myQueue的最前面。

```
Console.WriteLine(myQueue.Peek());
```

例6-19 队列操作(06-19.cs)。

```
using System;
using System.Collections;
class 06 - 19 - Queue
{
public static void Main()
    {
        Queue myQueue = new Queue();            //创建一个队列对象myQueue
        myQueue.Enqueue("Happy ");              //向队列中添加元素
        myQueue.Enqueue("New ");
        myQueue.Enqueue("Year ");
```

```
        foreach (string myString in myQueue)        //显示队列中的元素
        {
            Console.WriteLine("myString = " + myString);
        }
        int numElements = myQueue.Count;            //获得队列的元素个数

        for (int count = 0; count < numElements; count ++ );
        {
            //使用 Peek 方法查找队列中的下一项,然后使用 Dequeue 方法使其出列
            Console.WriteLine( "myQueue.Peek() = " + myQueue.Peek() );
            Console.WriteLine( "myQueue.Dequeue() = " + myQueue.Dequeue() );
        }
    }
}
```

程序的输出结果如下：

```
myString = Happy
myString = New
myString = Year
myQueue.Peek() = Happy
myQueue.Dequeue() = Happy
myQueue.Peek() = New
myQueue.Dequeue() = New
myQueue.Peek() = Year
myQueue.Dequeue() = Year
```

6.5.4 堆栈

堆栈(Stacks)是一种遵循“后进先出”(Last In First Out,LIFO)原则的数据集合,简称为栈。栈只能在一端输入输出,它有一个固定的栈底和一个浮动的栈顶。所有对堆栈的操作都是针对栈顶元素进行的。如果栈顶指针指向了栈底,说明当前的堆栈是空的。

创建一个堆栈对象,可以使用下述代码：

```
Stack myStack = new Stack();
```

对堆栈进行操作主要有如下方法：

(1) void Push(object item)：在堆栈顶部添加一个元素,也叫入栈。

(2) object Pop()：删除栈顶的元素,并返回该元素,也叫出栈。

(3) object Peek()：返回栈顶元素,但不删除它。

下面的例子演示了这个堆栈。

例 6-20 堆栈操作(06-20.cs)。

```
using System;
using System.Collections;
class 06 - 20 - Stacks
```

```
{
    public static void Main()
    {
        Stack myStack = new Stack();                // 创建一个堆栈对象 myStack
        myStack.Push ("Happy");                     // 向堆栈中添加元素
        myStack.Push ("New");
        myStack.Push ("Year");
        foreach (string myString in myStack)        //显示堆栈中的元素
        {
            Console.WriteLine("myString = " + myString );
        }
        int numElements = myStack.Count;            //获得堆栈中的元素个数
        for (int count = 0; count < numElements; count ++ );
        {
            //使用 Peek 方法找到堆栈中的下一个元素,然后使用 Pop 方法使其出栈
            Console.WriteLine( "myStack.Peek() = " + myStack.Peek() );
            Console.WriteLine( "myStack.Pop() = " + myStack.Pop() );
        }
    }
}
```

程序的输出结果如下：

```
myString = Year
myString = New
myString = Happy
myStack.Peek() = Year
myStack.Pop() = Year
myStack.Peek() = New
myStack.Pop() = New
myStack.Peek() = Happy
myStack.Pop() = Happy
```

6.6　习题与上机练习

1. 填空题

(1) C#提供一个默认的无参数构造函数，当实现了另一个有参数的构造函数，还想保留这个无参数的构造函数，应该写________个构造函数。

(2) 类中声明的属性往往具有 get() 和________两个访问器。

(3) 对于方法，参数传递分为值传递和________两种。

(4) 当在程序中执行到________语句时，将结束所在循环语句中循环体的一次执行。

2. 选择题

(1) 下列关于构造函数与析构函数的叙述中错误的是(　　)。

A. 均无返回值

B. 均不可定义为虚函数

C. 构造函数可以重载,而析构函数不可重载

D. 构造函数可带参数,而析构函数不可带参数

(2) 在类作用域中能够通过直接使用该类的(　　)成员名进行访问。

A. 私有　　B. 公用　　C. 保护　　D. 任何

(3) 类的以下特性中,可以用于方便地重用已有的代码和数据的是(　　)。

A. 多态　　B. 封装　　C. 继承　　D. 抽象

(4) 数据类型转换的类是(　　)。

A. Mod　　B. Convert　　C. Const　　D. Single

3. 问答题

(1) C#语言中,值类型和引用类型有何不同?

(2) 如何访问基类的函数?

(3) 构造函数有什么作用? 简述重载构造函数的好处。

(4) 结构和类的区别是什么?

4. 读程序题

(1) 写出以下程序的运行结果。

```
using System;
class Test
{
    public static void Main()
    {
        int x = 5;
        int y = x++;
        Console.WriteLine(y);
        y=++x;
        Console.WriteLine(y);
    }
}
```

(2) 写出下列函数的功能。

```
static float FH()
{
    float y=0,n=0;
    int x = Convert.ToInt32(Console.ReadLine());  //从键盘读入整型数据赋给 x
    while (x !=-1)
    {
        n++; y+=x;
        x = Convert.ToInt32(Console.ReadLine());
    }
if (n == 0)
    return y;
else
    return y/n;
}
```

5. 上机练习

(1) 编写一个学生类,用于处理学生信息(学号、姓名、性别、专业)。在创建学生类的实例时,把学生信息作为构造函数的参数输入,然后将学生信息在浏览器输出。

(2) 编写一个控制台应用程序,输出 1～5 的平方值,要求:

① 用 for 语句实现。

② 用 while 语句实现。

③ 用 do-while 语句实现。

(3) 编写一个类,输入矩形的长和宽,计算矩形的面积。

(4) 编写一个控制台应用程序,完成下列功能。

① 创建一个类,用无参数的构造函数输出该类的类名。

② 增加一个重载的构造函数,带有一个 string 类型的参数,在此构造函数中将传递的字符串打印出来。

③ 在 Main 方法中创建属于这个类的一个对象,不传递参数。

④ 在 Main 方法中创建属于这个类的另一个对象,传递一个字符串"This is a string."。

⑤ 在 Main 方法中声明类型为这个类的一个具有 5 个对象的数组,但不要实际创建分配到数组中的对象。

第 7 章　ASP.NET 服务器控件

为了提高 Web 开发的效率，ASP.NET 应用了“基于控件的可视化界面设计”和“事件驱动的程序运行模式”。通过使用 ASP.NET 提供的大量服务器控件，将这些控件拖放到 Web 窗体中，可以轻松地进行 ASP.NET 页面设计。同时，给特定的事件提供事件响应代码的编写模板，大大方便了 Web 软件开发者，提高了开发效率。

本章将介绍 Web 服务器控件的使用。

7.1　ASP.NET 页面的生命周期

ASP.NET 页面生命周期是 ASP.NET 中非常重要的概念，了解并掌握 ASP.NET 页面的生命周期，就能够在合适的生命周期内编写代码，执行事务，并开发高效的自定义控件。

ASP.NET 网页一般由两部分组成，即可视界面和处理逻辑。

- 可视界面：由 HTML 标记、ASP.NET 服务器控件等组成，即 aspx 文件。
- 处理逻辑：包含事件处理程序和代码，如 C# 代码，即 cs 文件。

ASP.NET 页面运行时，页面将经历一个生命周期，在生命周期内，该页面将执行一系列的步骤，包括控件的初始化，控件的实例化，还原状态和维护状态等，以及通过 IIS 反馈给用户呈现成 HTML。

一般来说，Web 页面的生命周期要经历如下阶段：

页面请求→开始→初始化→页面加载控件→验证→回发事件处理→呈现→卸载。

1. 页面请求(Page Request)

页面请求发生在 Web 页面生命周期开始之前。当用户请求一个 Web 页面时，ASP.NET 将确定是否需要分析或者编译该页面，或者是否可以在不运行页的情况下直接请求缓存响应客户端。

2. 开始(Start)

发生了请求后，页面就进入了开始阶段。在该阶段，页面将确定请求是回发请求还是新的客户端请求，并设置 IsPostBack 属性。

3. 初始化(Page Initialization)

在页面开始后，进入了初始化阶段。初始化期间，页面可以使用服务器控件，并为每个服务器控件进行初始化，即设置每一个控件的 UniqueID 属性。

4. 页面加载控件(Load)

如果当前请求是回发请求，则页面里各个控件的新的值和 ViewState 将被恢复或设置。

5. 验证(Validation)

在验证期间,页面中验证控件调用自己的 Validate 方法进行验证以便设置自己的 IsValid 属性,因为验证控件是在客户端和服务器端都要进行验证的。

6. 回发事件处理(Postback Event Handling)

如果请求是回发请求,则调用所有事件的处理程序。

7. 呈现(Rendering)

在呈现之前,会保存所有控件的 ViewState 视图状态。在呈现期间,页会调用每个控件的 Render 方法,将各个控件的 HTML 文本输出写到 Response 的 OutputStream 属性中。

8. 卸载(Unload)

完全呈现页面后,将页面发送到客户端。准备丢弃该页时,将调用卸载即卸载页面的属性并执行清理,资源被释放。

7.2 服务器控件概述

通常情况下,服务器控件都包含在 ASP.NET 页面中,能够被服务器端的程序代码访问和操作。

服务器控件都是 ASP.NET 页面上的对象,采用事件驱动的编程模型,客户端触发的事件在服务器端来处理。所有的服务器控件事件都传递两个参数,如按钮单击事件 Button_Click(object sender,EventArgs e)。其中,第一个参数 sender 表示引发事件的对象,以及包含任何事件特定信息的事件对象。第二个参数 e 是 EventArgs 类型,对于某些控件来说是特定于该控件的类型。

7.2.1 服务器控件的共有属性

共有属性就是所有的服务器控件都有的属性,这些属性主要用来设置控件的外观、布局、可访问性等,主要包括:

- 布局属性;
- 行为属性;
- 可访问属性;
- 数据属性;
- 外观属性。

每个服务器控件都有一个 id 属性和 runat="server"属性。其中,id 属性是服务器控件的唯一标识,供服务器端代码进行访问。

因此,定义一个服务器控件的基本语法为:

```
<asp:控件 id="控件标识" runat="server" 属性1=值1,…,属性n=值n />
```

服务器控件的属性既可以通过属性页窗口来设置(如图 7-1 所示),也可以通过 HTML 代码实现。

例如,一个包含服务器控件的 HTML 代码如下。

```
<form id="form1" runat="server">
    <asp:Button id="btnSubmit" Text="OK" OnClick="btnSubmit_Click" runat="server" />
    <asp:Label id="lblMessage" runat="server"/>
</form>
```

该页面包括两个服务器控件：asp:Button 按钮控件和 asp:Label 标签控件。其中，asp:Button 控件的 OnClick 属性声明了单击事件的处理程序名。该页面对应的程序代码如下。

```
protected void Page_Load(Object sender, EventArgs e)
{
    if (!IsPostBack)                                //判断页面是否第一次加载
    {
        lblMessage.Text = "页面第一次访问!";
    }
    else
    {
        lblMessage.Text = "页面被提交了!";
    }
}
void btnSubmit_Click (Object sender, EventArgs e)   //按下 OK 按钮后的处理代码
{
    btnSubmit.Text = "You click me! "
}
```

上述代码首先定义了一个 Page_Load 事件；其次定义了一个按钮的 Click 事件处理程序 btnSubmit_Click。

当页面初次被加载时，会执行 Page_Load 中的代码，通过判断 Page 对象的 IsPostBack 属性来确定页面是否第一次加载。

可以看出，包含 Web 服务器控件的代码分成两部分，一部分是 HTML 代码；另一部分是程序代码。其代码的文件存储形式有两种，一种是全部保存在一个 aspx 文件里；另一种是分成两个文件，即页面代码 aspx 文件和程序代码 cs 文件。

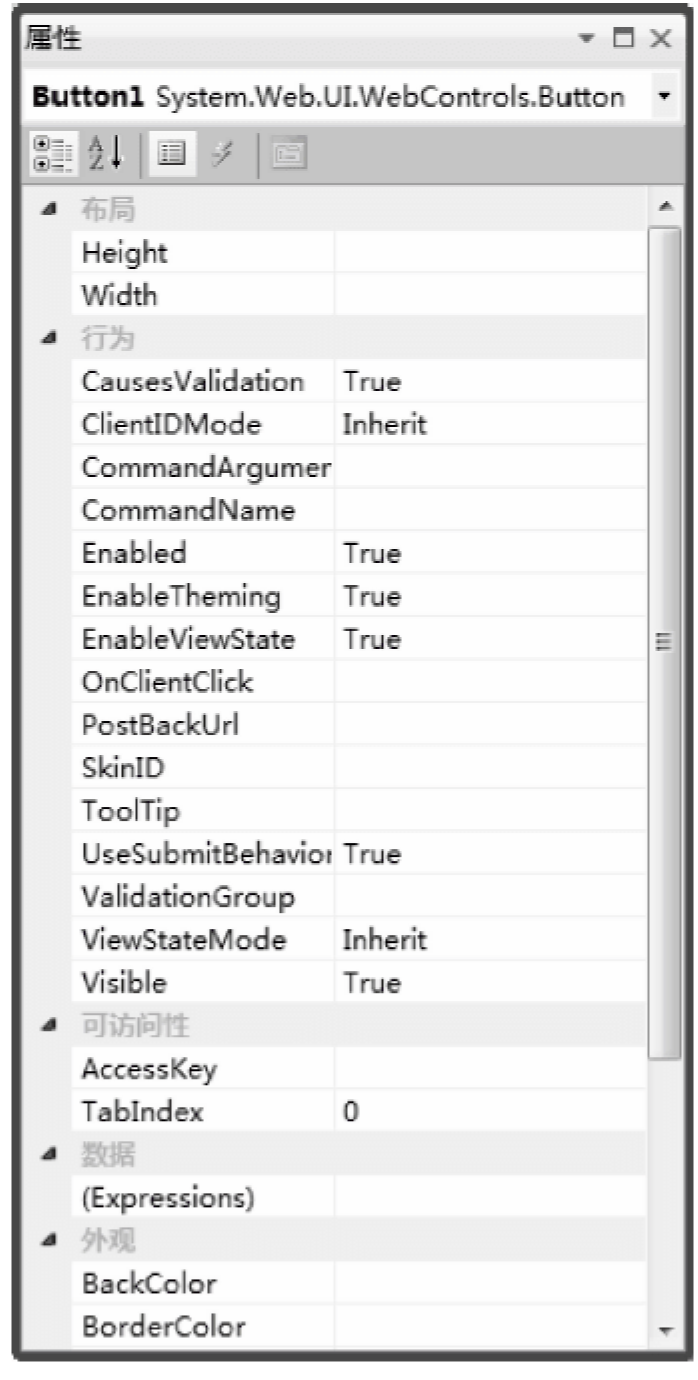


图 7-1　服务器控件的属性页

7.2.2　服务器控件的共有事件

服务器控件的事件用于当服务器进行到某个时刻引发从而完成某些任务。事件的回发会导致页面的 Init 事件和 load 事件等。在页面的 Onload 事件方法中编写代码时，还需要根据情况判断是否需要检测回发事件，常用的检测方法就是通过 Page.isPostBack、Page.IsCallback、Page.IsCrossPagePostBack 等属性来判断页面事件的状态。服务器控件共有的事件如表 7-1 所示。

表 7-1　服务器控件共有的事件

事　　件	说　　明
DataBinding	当控件上的 DataBind 方法被调用,并且该控件被绑定到一个数据源时被激发
Disposed	从内存中释放一个控件时激发
Init	控件被初始化时激发
Load	把控件装入页面时会激发,该事件在 Init 后发生
PreRender	控件准备生成它的内容时激发
Unload	从内存中卸载控件时激发

7.2.3　服务器控件的分类

ASP.NET 服务器控件主要分为 HTML 控件、Web 服务器控件、验证控件和用户控件 4 类,说明如下:

(1) HTML 服务器控件(HTML Server Controls):这是对 HTML 标记的扩展,每个 HTML 控件都和原来的 HTML 标记一一对应。通常,ASP.NET 文件中的 HTML 元素默认作为文本进行处理。为了使这些元素可编程化,需要添加 runat="server" 属性,指示 HTML 元素应作为服务器控件进行处理。

(2) Web 服务器控件(Web Server Controls):Web 服务器控件是服务器可理解的特殊 ASP.NET 标签。它的功能更加强大、使用更加灵活,但不一定对应到某个 HTML 元素。

(3) 验证控件(Validation Controls):用于验证用户输入。如果没有通过验证,将向用户显示一条错误消息。一般与 HTML 控件或者 Web 控件结合使用。

(4) 用户控件(User Controls):由用户创建,可以嵌入到 Web 窗体中的控件。

7.3　标准的 Web 服务器控件

Web 服务器控件位于 System. Web. UI. WebControls 命名空间中,是从 WebControl 基类直接或间接派生出来的。

Web 服务器控件的标准控件如图 7-2 所示。它通常分为 4 类控件:

- 文本输入与显示控件;
- 控制权转移控件;
- 选择控件;
- 容器控件。

下面分别对这 4 类控件进行介绍。

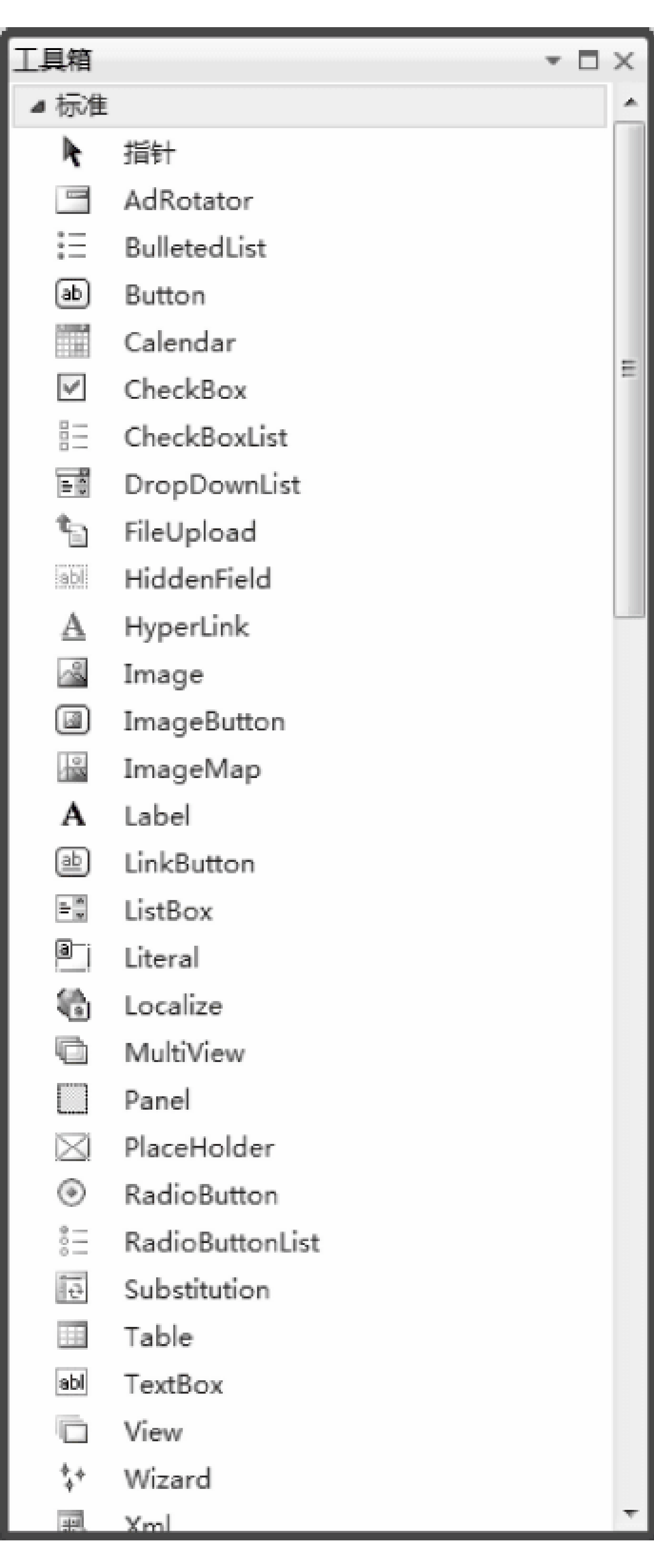


图 7-2　Web 服务器控件

7.3.1 文本输入与显示控件

文本输入控件包括文本控件(TextBox)。

显示控件包括显示文本的标签控件(Label)和显示图片的图像控件(Image)两种。

1. 文本控件(TextBox)

TextBox 控件用于提供文本编辑能力。TextBox 控件支持多种模式,可以用来实现单行输入、多行输入和密码输入。表 7-2 所示为文本控件的属性列表。

表 7-2 TextBox 控件的属性

属　性	说　明
AutoPostBack	在文本修改以后,是否自动重传。默认为 False,当设置为 True 时,用户更改内容后触发 TextChanged 事件
Columns	文本框的宽度
EnableViewState	控件是否自动保存其状态以用于往返过程
MaxLength	用户输入的最大字符数
ReadOnly	是否为只读
Rows	作为多行文本框时所显示的行数
Text	获取或设置 TextBox 控件中的数据
TextMode	显示模式,取值为 SingleLine、MultiLine 或 Password

例如:

```
private void txtUserName_TextChanged(object sender, System.EventArgs e)
{
    Label1.Text = txtUserName.Text;
}
```

2. 标签控件(Label)

Label 控件用于在页面中显示只读的静态文本或数据绑定的文本。当触发事件时,某一段文本能够在运行时更改。示例代码如下。

```
<asp:Label ID = "Label1" runat = "server" Text = "Hello"></asp:Label>
```

上述代码声明了一个标签控件,并将该控件的 ID 属性设置为默认值 Label1。由于该控件是服务器端控件,所以包含 runat="server" 属性。表 7-3 所示为标签控件的属性和事件。

表 7-3 Label 控件的属性和事件

属性/事件	说　明
Text 属性	获取或设置 Label 控件中的数据
TextChanged 事件	用户输入信息后离开 TextBox 控件时引发的事件

3. 图像控件(Image)

图像控件用来在 Web 窗体中显示图片或图像,图像控件常用的属性如表 7-4 所示。

表 7-4　Image 控件的属性

属　　性	说　　明
AlternateText	在图像无法显示时显示的替换文字
DescriptionUrl	包含更详细的图像说明的 URL
GenerateEmptyAlternateText	当未指定替换文字时，是否生成空的替换文字属性，默认为 False
ImageAlign	图像的对齐方式
ImageUrl	要显示图像的 URL
ToolTip	把鼠标放在控件上时显示的工具提示

当图片无法显示的时候，图片将被替换成 AlternateText 属性中的文字，ImageAlign 属性用来控制图片的对齐方式，而 ImageUrl 属性用来设置图像连接地址。图像控件具有可控性的优点，可以通过编写 HTML 来控制图像控件。例如，图像控件声明代码如下。

```
<asp:Image ID = "Image1" runat = "server" AlternateText = "图片连接失效"
ImageUrl = "http://www.shangducms.com/images/cms.jpg" />
```

上述代码设置了一个图片，当图片失效的时候提示图片连接失效。

注意：当双击图像控件时，系统并没有生成事件所需要的代码段，说明 Image 控件不支持任何事件。

7.3.2　控制权转移控件

控制权转移控件包括 4 种类型：

(1) Button 控件：显示标准 HTML 窗体按钮。

(2) LinkButton 控件：在按钮上显示超文本链接。

(3) ImageButton 控件：显示图像按钮。

(4) Hyperlink 控件：在某些文本上显示超文本链接。

1. 按钮控件

Button、LinkButton 和 ImageButton 为按钮控件，能够触发事件，或将网页中的信息回传给服务器。它们的作用基本相同，主要是表现形式不同，如图 7-3 所示。其声明代码如下：



图 7-3　三种按钮类型(Button、LinkButton 和 ImageButton)

```
<asp:Button ID="Button1" runat="server" Text="click me" />  <br />
<asp:LinkButton ID="LinkButton1" runat="server">click me</asp:LinkButton><br />
<asp:ImageButton ID="ImageButton1" runat="server" ImageURL="a.bmp" />
```

按钮控件用于事件的提交，通常包含一些公共的属性和事件，表 7-5 和表 7-6 分别所示为按钮控件的公共属性和特殊属性，表 7-7 所示为它们的公共事件。

表 7-5　按钮控件（Button、LinkButton 和 ImageButton）的公共属性

属　　性	说　　明
CausesValidation 属性	按钮是否导致激发验证，默认为 True
CommandArgument 属性	与此按钮关联的命令参数
CommandName 属性	与此按钮关联的命令
Enabled 属性	控件的已启用状态，默认为 True
EnableViewState 属性	控件是否自动保存其状态以用于往返过程，默认为 TRUE
OnClientClick 属性	在客户端 OnClick 上执行的客户端脚本
ValidationGroup 属性	当控件导致回发时应验证的组
ViewStateMode 属性	确定该按钮是否启用了 viewstate（默认为从父代继承）

表 7-6　Button、ImageButton 和 LinkButton 的特殊属性

控件名称	属　　性	说　　明
Button	UseSubmitBehavior 属性	指示按钮是否呈现为提交按钮
	Text 属性	在按钮上显示的文本
ImageButton	ImageAlign 属性	图像的对齐方式
	PostBackURL 属性	单击按钮时所发送到的 URL
	AlternateText 属性	在图像无法显示时显示的替换文字
	ImageURL 属性	要显示的图像的 URL
LinkButton	Text 属性	要为该链接显示的文本
	PostBackURL 属性	单击按钮时所发送到的 URL

表 7-7　按钮控件（Button、LinkButton 和 ImageButton）的公共事件

事　　件	说　　明
Click 事件	单击按钮时会引发该事件
Command 事件	在单击按钮并定义关联的命令时激发
DataBinding 事件	在要计算控件的数据绑定表达式时激发
Disposed 事件	在控件已被释放后激发
Init 事件	在初始化页后激发
Load 事件	在加载页后激发
PreRender 事件	在呈现该页前激发
Unload 事件	在卸载该页时激发

值得一提的是，最常用的按钮事件是 Click 单击和 Command 命令事件。Click 事件不能传递参数，处理的事件相对简单。Command 事件可以传递参数，负责传递参数的是 CommandArgument 属性和 CommandName 属性。

当按钮同时包含 Click 事件和 Command 事件时，通常情况下会执行 Command 事件。通过判断按钮的 CommandArgument 属性和 CommandName 属性值，可以执行不同的方法。这样就实现了同一个按钮根据不同的值进行不同的处理和响应，或者多个按钮与一个处理代码相关联。相比 Click 单击事件而言，Command 命令事件具有更高的可控性。

2. 超链接控件(HyperLink)

超链接控件相当于实现了 HTML 代码中的" "效果。当拖动一个超链接控件到页面时，系统会自动生成控件声明代码，示例代码如下所示。

```
<asp:HyperLink ID="HyperLink1" runat="server">HyperLink</asp:HyperLink>
```

表 7-8 所示是 HyperLink 控件的属性。

表 7-8　HyperLink 的属性

属　性	说　明	属　性	说　明
Text 属性	要为该链接显示的文本	NavigateURL 属性	定位到的 URL
ImageURL 属性	要显示的图像的 URL	Target 属性	NavigateUrl 的目标框架

7.3.3　选择控件

顾名思义，选择控件就是在一组选项中选出一项或多项，通常包括 4 大类型：

(1) 单选控件(RadioButton)：用于在一个选项列表中选择一个选项，使用时通常会与其他 RadioButton 控件组成一组，以提供一组互斥的选项。

(2) 复选框控件(CheckBox)：可选择一项或多项，每个选项可在选中和清除这两种状态间切换。

(3) 下拉列表控件(DropdownList)：允许用户从预定义的列表中选择一项。

(4) 列表控件(ListBox)：允许用户从预定义列表中选择一项或多项。

下面分别对这几种控件加以介绍。

1. 单选控件和单选组控件(RadioButton 和 RadioButtonList)

1) 单选控件(RadioButton)

单选控件可以为用户选择某一个选项，单选控件的常用属性和事件如表 7-9 所示。

表 7-9　RadioButton 的属性和事件

属性/事件	说　明
AutoPostBack 属性	当单击控件时，自动回发到服务器，默认为 False
CausesValidation 属性	该控件是否导致激发验证，默认为 False
Checked 属性	控件的已选中状态，默认为 False
GroupName 属性	此单选控件所属的组名
Text 属性	显示的文本标签
TextAlign 属性	文本标签相对于控件的对齐方式，默认为 Right
CheckedChanged 事件	在更改控件的选中状态时激发

单选控件通常需要 Checked 属性来判断某个选项是否被选中，多个单选控件之间可能存在着某些联系，这些联系通过 GroupName 进行约束和联系，示例代码如下所示。

```
<form id = "form1" runat = "server">
<asp:RadioButton ID = "RadioButton1" runat = "server" GroupName = "chos" Text = "Choose1"
AutoPostBack = "true"/>
<asp:RadioButton ID = "RadioButton2" runat = "server" GroupName = "chos" Text = "Choose2"
AutoPostBack = "true"/>
<asp:RadioButton ID = "RadioButton3" runat = "server" GroupName = "chos" Text = "Choose3"
AutoPostBack = "true"/>
</form>
```

上述代码声明了三个单选控件，并将 GroupName 属性都设置为 chos。单选控件中最常用的事件是 CheckedChanged，当控件的选中状态改变时，则触发该事件，示例代码如下。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void RadioButton1_CheckedChanged(object sender, EventArgs e)
    {
        Label1.Text = "第一项被选中";
    }
    protected void RadioButton2_CheckedChanged(object sender, EventArgs e)
    {
        Label1.Text = "第二项被选中";
    }
    protected void RadioButton3_CheckedChanged(object sender, EventArgs e)
    {
        Label1.Text = "第三项被选中";
    }
}
```

上述代码中，当选中状态被改变时，则触发相应的事件，显示被选中的项，如图 7-4 所示。

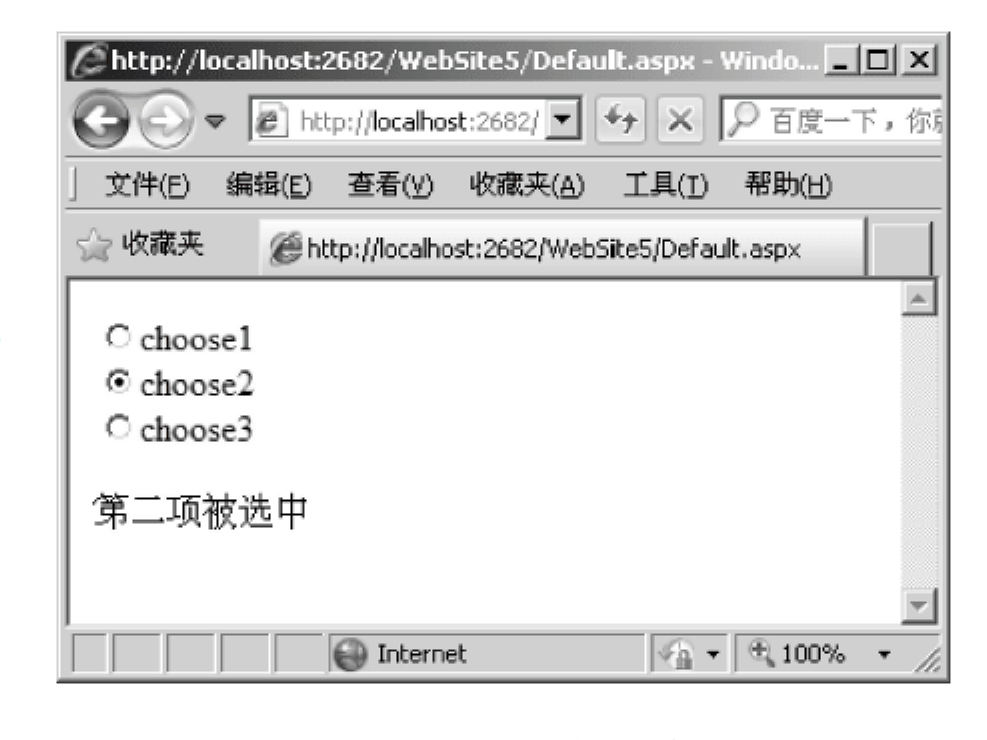


图 7-4　单选控件的使用

与 TextBox 文本框控件相同的是，单选控件不会自动进行页面回传，必须将 AutoPostBack 属性设置为 true 时才能在焦点丢失时触发相应的 CheckedChanged 事件。

2）单选组控件（RadioButtonList）

单选组控件也是只能选择一个项目的控件，而与单选控件不同的是，单选组控件没有

GroupName 属性,但是却能够列出多个单选项目。另外,单选组控件所生成的代码也比单选控件实现的相对较少。单选组控件添加项如图 7-5 所示。

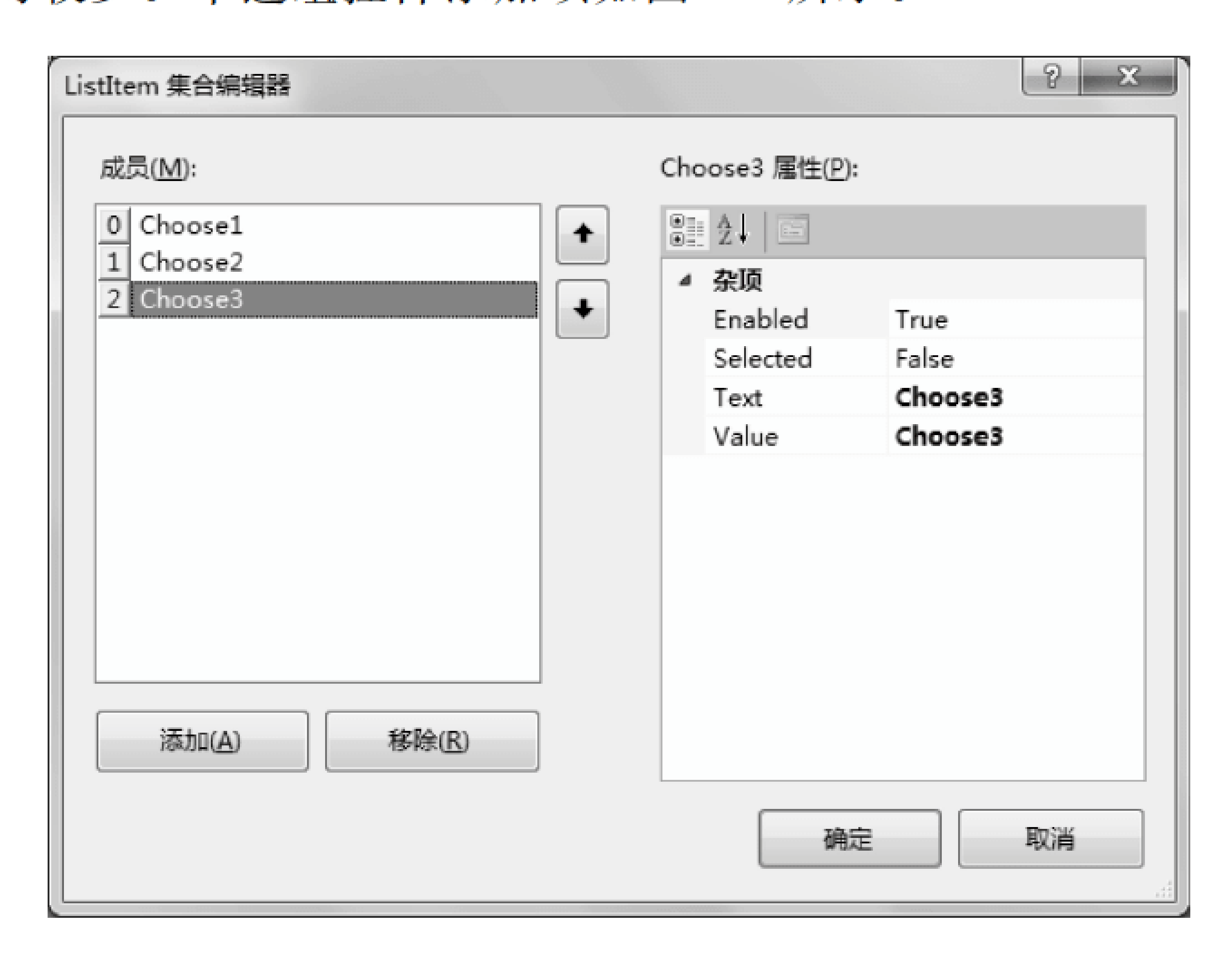


图 7-5　单选组控件添加项

添加项目成员后,系统自动在.aspx 页面声明服务器控件代码,代码如下所示。

```
<asp:RadioButtonList ID = "RadioButtonList1" runat = "server"
        SelectedIndexChanged = "RadioButtonList1_SelectedIndexChanged">
        <asp:ListItem>Choose1</asp:ListItem>
        <asp:ListItem>Choose2</asp:ListItem>
        <asp:ListItem>Choose3</asp:ListItem>
</asp:RadioButtonList>
```

上述代码使用了单选组控件进行单选功能的实现。单选组控件的属性和事件如表 7-10 所示。

表 7-10　RadioButtonList 的属性和事件

属性/事件	说　明
AutoPostBack 属性	当选定内容更改后,自动回发到服务器,默认为 False
DataMember 属性	用于绑定的表或视图
DataSourceID 属性	将被用作数据源的 DataSource 的控件 ID
DataTextFiled 属性	数据源中提供项文本的字段
DataTextFormatString 属性	应用于文本字段的格式,如,"{0:d}"
DataValueField 属性	数据源中提供项值的字段
Items 属性	列表中项的集合
RepeatColumns 属性	用于布局项的列数,初值为 0
RepeatDirection 属性	项的布局方向,默认为 Vertical
RepeatLayout 属性	项是否在某个表或者流中重复
SelectedIndexChanged 事件	在更改选定索引后激发
TextChanged 事件	在更改文本属性后激发

同单选控件一样，双击单选组控件时，系统会自动生成 SelectedIndexChanged 事件的声明，可以在该事件中编写代码。当选定一项内容时，示例代码如下所示。

```
protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e)
{
    Label1.Text = RadioButtonList1.Text;            //文本标签的值等于选择的控件的值
}
```

2. 复选框控件和复选组控件（CheckBox 和 CheckBoxList）

1）复选框控件（CheckBox）

同单选框控件一样，复选框也是通过 Check 属性判断是否被选中。不同的是，复选框控件没有 GroupName 属性，以下代码声明了两个复选框控件。

```
<form id="form2" runat="server">
<asp:CheckBox ID="CheckBox1" runat="server" Text="Check1" AutoPostBack="true" />
<asp:CheckBox ID="CheckBox2" runat="server" Text="Check2" AutoPostBack="true" />
</form>
```

当双击复选框控件时，系统会自动生成 CheckedChanged 事件的声明。当复选框控件的选中状态被改变后，会激发该事件。示例代码如下所示。

```
protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
{
    Label1.Text = "选框 1 被选中";            //当选框 1 被选中时
}
protected void CheckBox2_CheckedChanged(object sender, EventArgs e)
{
    Label1.Text = "选框 2 被选中";            //当选框 2 被选中时
    Label1.Font.Size = FontUnit.XXLarge;
}
```

上述代码分别为两个选框设置了事件，设置了当选择选框 1 时，则文本标签输出“选框 1 被选中”。当选择选框 2 时，则输出“选框 2 被选中”。

对于复选框而言，用户可以在复选框控件中选择多个选项，所以就没有必要为复选框控件进行分组。也就是说，复选框控件没有 GroupName 属性。

2）复选组控件（CheckBoxList）

同单选组控件相同，.NET 服务器控件中同样包括了复选组控件（CheckBoxList），拖动一个复选组控件到页面可以添加复选组列表。添加在页面后，系统生成代码如下。

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server" AutoPostBack="True"
          SelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
          <asp:ListItem Value="Choose1">Choose1</asp:ListItem>
          <asp:ListItem Value="Choose2">Choose2</asp:ListItem>
          <asp:ListItem Value="Choose3">Choose3</asp:ListItem>
</asp:CheckBoxList>
```

复选组控件最常用的是 SelectedIndexChanged 事件。当控件中某项的选中状态被改变

时，则会触发该事件。示例代码如下。

```
protected void CheckBoxList1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (CheckBoxList1.Items[0].Selected)        //判断某项是否被选中
    {
        Label1.Font.Size = FontUnit.XXLarge;    //更改字体大小
    }
    if (CheckBoxList1.Items[1].Selected)        //判断是否被选中
    {
        Label1.Font.Size = FontUnit.XLarge;     //更改字体大小
    }
    if (CheckBoxList1.Items[2].Selected)
    {
        Label1.Font.Size = FontUnit.XSmall;
    }
}
```

上述代码中，Item数组是复选组控件中项目的集合，其中Items[0]是复选组中的第一个项目。CheckBoxList1.Items[0].Selected用来判断是否被选中。上述代码用来修改Label标签的字体大小。

注意：复选组控件与单选组控件不同的是，不能够直接获取复选组控件某个选中项目的值，因为复选组控件返回的是第一个选择项的返回值，只能够通过Item集合来获取选择某个或多个选中的项目值。

3. 下拉列表控件（DropDownList）

列表控件能够在一个控件中为用户提供多个选项，既简化用户的输入，同时又防止用户输入错误的选项。列表控件主要包括下拉列表DropDownList和多项选择列表ListBox两种。

使用DropDownList下拉列表控件，可以有效地避免用户输入无效或错误的信息。例如，当输入性别时，除了男就是女，输入其他的信息就是错误的。下列语句声明了一个DropDownList列表控件。

```
<asp:DropDownList ID = "DropDownList1" runat = "server" AutoPostBack = "True"
     SelectedIndexChanged = "List_Changed" >
          <asp:ListItem > 1 </asp:ListItem >
          <asp:ListItem > 2 </asp:ListItem >
          <asp:ListItem > 3 </asp:ListItem >
          <asp:ListItem > 4 </asp:ListItem >
</asp:DropDownList >
```

DropDownList控件也可以绑定数据源控件，常用的事件是SelectedIndexChanged，当用户选择相应的项目，使得DropDownList控件选择项发生变化时，则会触发该事件，示例代码如下。

```
protected void DropDownList1_SelectedIndexChanged1(object sender, EventArgs e)
{
```

```
        Label1.Text = "你选择了第" + DropDownList1.Text + "项";
    }
```

下拉列表控件的属性和事件如表 7-11 所示。

表 7-11　DropDownList 的属性和事件

属性/事件	说　明
AppendDataBoundItems 属性	将数据绑定项追加到静态声明的列表项上,默认为 False
AutoPostBack 属性	当选定内容更改后,自动回发到服务器,默认为 False
DataMember 属性	用于绑定的表或视图
DataSourceID 属性	将被用作数据源的 DataSource 的控件 ID
DataTextFiled 属性	数据源中提供项文本的字段
DataTextFormatString 属性	应用于文本字段的格式,如"{0:d}"
DataValueField 属性	数据源中提供项值的字段
Items 属性	列表中项的集合
SelectedIndexChanged 事件	在更改选定索引后激发
TextChanged 事件	在更改文本属性后激发

4. ListBox 列表控件

相对于 DropDownList 控件而言,ListBox 控件可以通过 SelectionMode 属性指定用户是否允许多项选择。当创建一个 ListBox 列表控件后,示例代码如下。

```
<asp:ListBox ID = "ListBox1" runat = "server" AutoPostBack = "True"
        onselectedindexchanged = "ListBox1_SelectedIndexChanged">
        <asp:ListItem>第 1 项</asp:ListItem>
        <asp:ListItem>第 2 项</asp:ListItem>
        <asp:ListItem>第 3 项</asp:ListItem>
        <asp:ListItem>第 4 项</asp:ListItem>
</asp:ListBox>
<asp:Label ID = "Label1" runat = "server" Text = " 你所选的项目为: "></asp:Label>
```

ListBox 控件的属性与 DropDownList 控件基本上相同,只增加了两个属性,如表 7-12 所示。

表 7-12　ListBox 控件比 DropDownList 控件增加的属性

属　性	说　明
Rows	要显示的可见行的数目
SelectionMode	列表的选择模式,默认为 Single

设置 SelectionMode 属性为 Single 时,表明只允许用户从列表框中选择一个项目;如果设置 SelectionMode 属性为 Multiple 时,用户可以按住 Ctrl 键或使用 Shift 组合键,从列表中选择多个数据项。

同样,SelectedIndexChanged 也是 ListBox 控件中最常用的事件,可以对事件编码如下。

```
protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    Label1.Text = "你选择了" + ListBox1.Text ;
}
```

上面的程序实现了与 DropDownList 同样的效果。

当用户需要选择 ListBox 列表中的多项时，即 SelectionMode 属性为 Multiple，开发人员编写的事件代码如下所示。

```
protected void ListBox1_SelectedIndexChanged1(object sender, EventArgs e)
{
    Label1.Text += "<br>你选择了" + ListBox1.Text ;
}
```

上述代码使用了"＋＝"运算符，当用户每多选一项的时候，都会触发 SelectedIndexChanged 事件，如图 7-6 所示。



图 7-6　ListBox 控件的多选效果

7.3.4　容器控件

有两种类型的容器控件：

- 面板控件(Panel)：可用作静态文本和其他控件的父级控件。
- 占位控件(PlaceHolder)：存储动态添加到网页上的服务器控件的容器。

下面对这两个控件加以介绍。

1. 面板控件

面板控件可以作为一组控件的容器。通过设置在面板控件内的所有控件是显示还是隐藏，从而达到设计者的特殊目的。当创建一个面板控件时，系统生成的 HTML 代码如下。

```
<asp:Panel ID = "Panel1" runat = "server">
</asp:Panel>
```

面板控件的常用功能就是显示或隐藏一组控件，其 Visible 属性的默认值为 True 。设置 Panel 控件的 HTML 代码如下。

```
<form id = "form1" runat = "server">
    <asp:Button ID = "Button1" runat = "server" Text = "Show" />
    <asp:Panel ID = "Panel1" runat = "server" Visible = "False">
        <br /> The controls in a Panel are in follows.
        <br />
        <asp:Label ID = "Label1" runat = "server" Text = "Hello"></asp:Label>
        <asp:TextBox ID = "TextBox1" runat = "server"></asp:TextBox>
    </asp:Panel>
</form>
```

上述代码创建了一个 Panel 控件，初始状态为不可见。在 Panel 控件外有一个 Button 控件。当用户单击 Button 控件时，将显示 Panel 控件。cs 代码如下所示。

```
protected void Button1_Click(object sender, EventArgs e)
{
    Panel1.Visible = true;              //Panel 控件显示可见
}
```

当页面初次被载入时，Panel 控件以及 Panel 控件内的全部控件都为隐藏，如图 7-7 所示。当用户单击 Button 时，则 Panel 控件及其内部的控件都为可见，如图 7-8 所示。

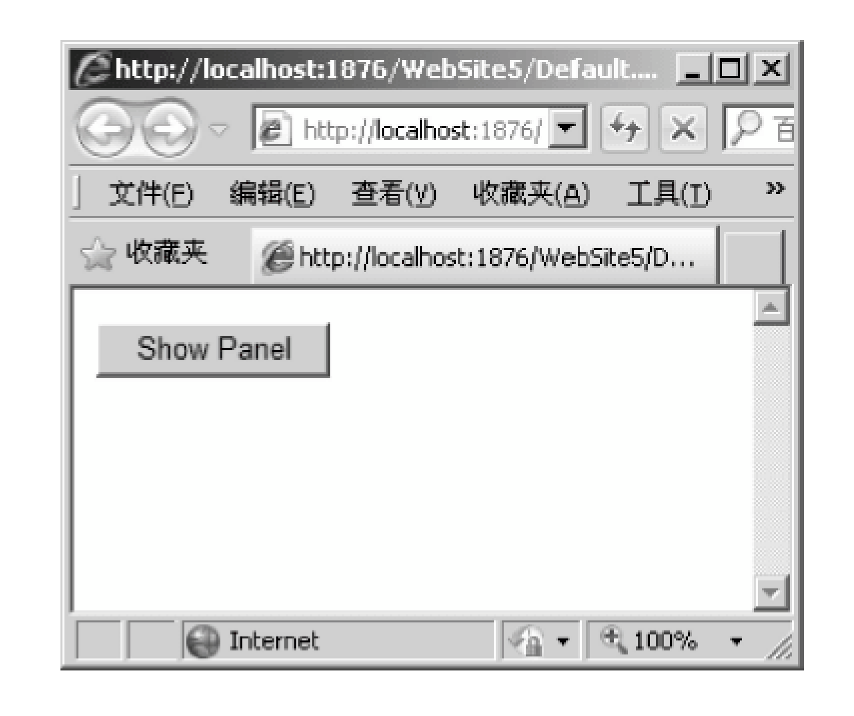


图 7-7　Panel 控件隐藏



图 7-8　Panel 控件被显示

Panel 控件还包含一个 GroupText 属性，当 Panel 控件的 GroupText 属性被设置时，Panel 将会被创建一个带标题的分组框，效果如图 7-9 所示。

2. 占位控件

与面板控件相同的是，占位控件 PlaceHolder 也是控件的容器，但是在 HTML 页面呈现中本身并不产生 HTML，创建一个 PlaceHolder 控件代码如下。

```
<asp:PlaceHolder ID = "PlaceHolder1" runat = "server">
</asp:PlaceHolder>
```

在 CS 页面中，允许用户动态地在 PlaceHolder 上创建控件，CS 页面代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    TextBox text = new TextBox();                   //创建一个 TextBox 对象
    text.Text = "happy";
    this.PlaceHolder1.Controls.Add(text);           //为占位控件动态增加一个控件
}
```

上述代码动态创建了一个 TextBox 控件并显示在占位控件中，运行效果如图 7-10 所示。

开发人员不仅能够通过编程在 PlaceHolder 控件中添加控件，同样可以在 PlaceHolder 控件中拖动相应的服务器控件进行控件呈现和分组。

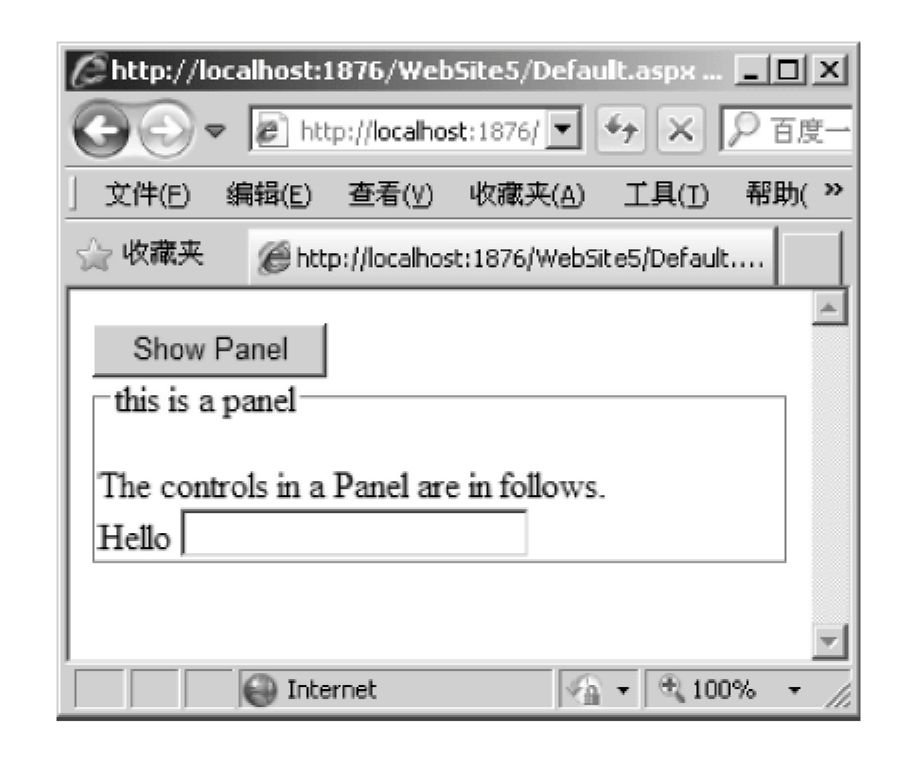


图 7-9 Panel 控件的 GroupText 属性

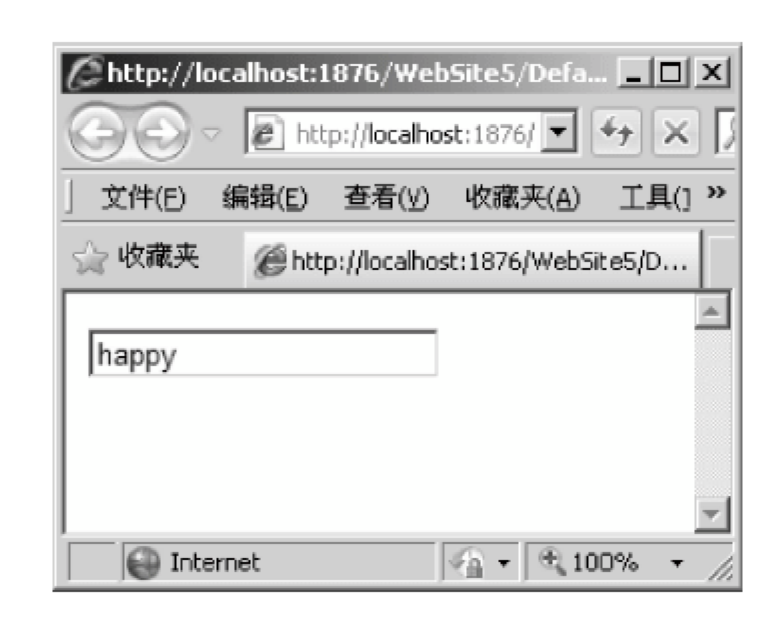


图 7-10 PlaceHolder 控件的使用

7.4 验 证 控 件

Visual Studio 2010 提供了强大的数据验证控件，可以验证用户的输入，并在验证失败的情况下显示错误消息。在 Visual Studio 的工具箱中见到的验证控件如图 7-11 所示。

注意：验证控件本身并不接受用户的输入，需要与其他控件（如 TextBox）相配合完成验证数据的工作，可以使用验证控件的 ControlToValidate 属性将验证控件与被验证控件关联起来。每个验证控件的基本说明如表 7-13 所示。

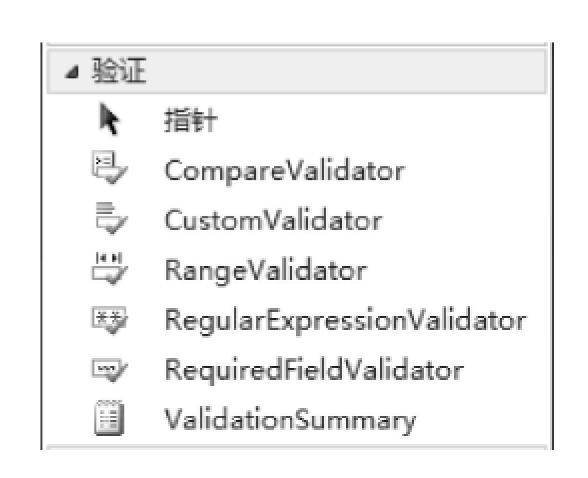


图 7-11 工具箱中的验证控件

表 7-13 验证控件的使用说明

验 证 控 件	功 能 说 明
RequiredFieldValidator	确保用户不跳过输入
CompareValidator	使用比较运算符（大于、小于、等于）将输入控件与一个固定值或另一个输入控件进行比较
RangeValidator	与 CompareValidator 非常相似，用来检查输入是否在两个值或其他输入控件的值之间
RegularExpressionValidator	检查用户的输入是否与正则表达式定义的模式相匹配；允许检查可预知的字符序列，如电话号码、邮政编码、社会保障号等
CustomValidator	允许用户编写自己的验证逻辑以检查用户的输入，通常用于奇偶验证
ValidationSummary	验证总结，以摘要的形式显示页上所有验证程序的验证错误

7.4.1 必须输入验证控件

在实际的应用中，如在用户填写表单时，有一些项目是必填项，如用户名和密码。使用必须输入验证控件（RequiredFieldValidator）能够要求用户在特定的控件中必须提供相应的信息，否则就提示错误信息。RequiredFieldValidator 控件的格式如下：

```
<asp:RequiredFieldValidator id = "控件名称" runat = "server"
```

```
ControlToValidate = "要检查的控件名称"
ErrorMessage = "出错信息" Display = "Dynamic | Static | None"  />
```

示例代码如下。

```
<form id = "form1" runat = "server">
<div>用户名:
<asp:TextBox ID = "txtName" runat = "server"></asp:TextBox>
<asp:RequiredFieldValidator ID = "Validator1" runat = "server"
  ControlToValidate = " txtName" ErrorMessage = "用户名不能为空" Display = "Static">
</asp:RequiredFieldValidator><br />
  密  码:<asp:TextBox ID = "txtPass" runat = "server"></asp:TextBox><br />
<asp:Button ID = "Validate1" runat = "server" Text = "OK" /><br />
</div>
</form>
```

上述代码中,RequiredFieldValidator 控件通过它的 ControlToValidate 属性绑定了一个文本控件 txtName(要验证的控件),当输入值为空且单击提交按钮时,则提示错误信息"用户名不能为空",如图 7-12 所示。此时,用户的所有页面输入都不会提交。只有将必填输入项都填写完成,页面才会向服务器提交数据。

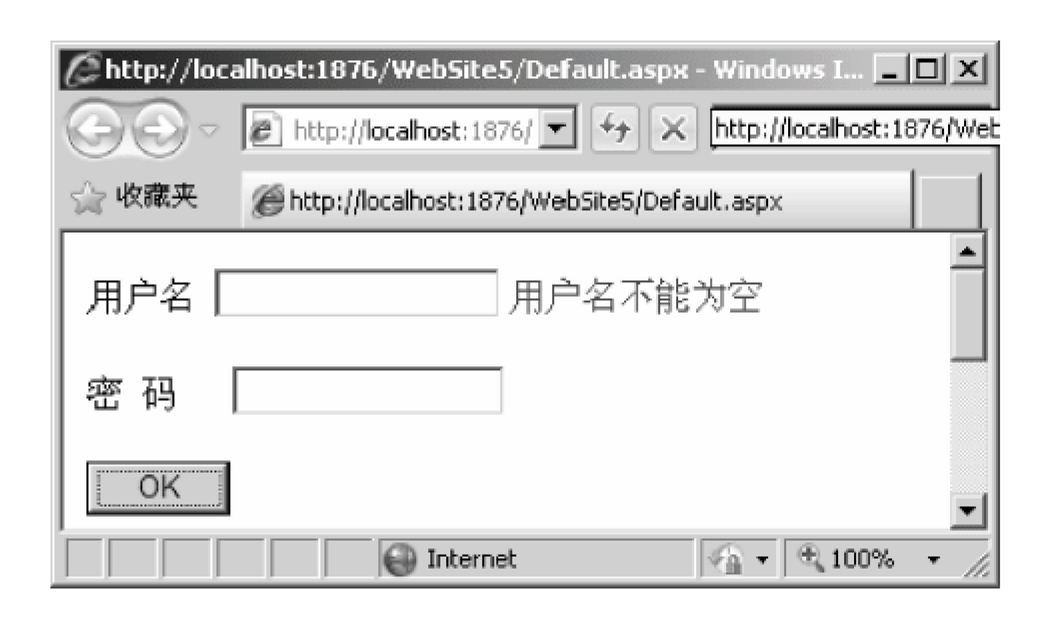


图 7-12　RequiredFieldValidator 验证控件

值得注意的是,RequiredFieldValidator 控件的 Initialvalue 属性(表示要验证的字段的初始值)默认值为空串。因此,当用户什么都不输入而直接单击提交按钮时将显示出错。仅当输入控件失去焦点,而且用户在此输入控件中输入的值等于 Initialvalue 属性值时,RequiredFieldValidator 控件才认为其数据不能通过验证。

7.4.2　比较验证控件

比较验证控件(CompareValidator)可以对比在两个控件中输入的数据。例如,在修改密码时,通常需要在两个文本框中分别输入一次新密码,并将两次输入的密码进行比对。

CompareValidator 控件的格式如下:

```
<asp:CompareValidator id = "控件名称"
ControlToValidate  = "要验证的控件 ID"
ControlToCompare  = "要比较的控件 ID"
Type  = "String | Integer | Date | Double| Currency"
```

```
Operator = "Equal|NotEqual|
        GreaterThan|GreaterThanEqual|LessThan|LessThanEqual|DataTypeCheck"
ErrorMessage = "出错信息"  Display = "Dynamic | Static | None"
runat = "server" />
```

CompareValidator 控件的属性如表 7-14 所示。

表 7-14　比较验证控件的属性

属　　性	说　　明
ControlToValidate	要验证的控件 ID
ControlToCompare	用于进行比较的控件 ID
Type	表示要比较的两个值的数据类型，取值有 String、Integer、Date、Double、Currency
Operator	表示要使用的比较运算符
ErrorMessage	出错提示信息
Text	当验证的控件无效时显示的验证程序文本
Display	验证程序的显示方式。取值包括三种 Dynamic：不出错的时候该控件不占用页面位置 Static：不出错的时候该控件占用页面位置 None：不显示出错信息
SetFocusOnError	控件无效时，验证程序是否在控件上设置焦点。默认为 False
ValueToCompare	用于进行比较的值

注意：可以直接将与 CompareValidator 控件相关联的输入控件的值与某个特定值进行比较，只需将 CompareValidator 控件的 ValueToCompare 属性设定为要比较的特定值即可。在这种情况下，不需要另外指定 ControlToCompare 属性。

CompareValidator 控件的示例代码如下。

```
<form id = "form2" runat = "server">
<div >密码 1:
<asp:TextBox id = "passwd1" TextMode = "Password" runat = "server" />
<br />密码 2:
<asp:TextBox id = "passwd2" TextMode = "Password" runat = "server" />
<asp:CompareValidator id = "Validator2" runat = "server"
    ControlToValidate = "passwd1" ControlToCompare = "passwd2"
    Type = "String" Operator = "Equal"
    Display = "static" ErrorMessage = "两者不一致">
</asp:CompareValidator >
<br /><asp:Button id = "Validate2" runat = "server" text = " 验 证 " />
</form >
```

上述代码中，判断两个密码输入框 passwd1 和 passwd2 中的输入值是否一致，比较类型为 String，比较运算符为 Equal，如果不相等则提示出错，如图 7-13 所示。

7.4.3　范围验证控件

范围验证控件(RangeValidator)可以要求用户输入特定范围内的数据。该控件可以检

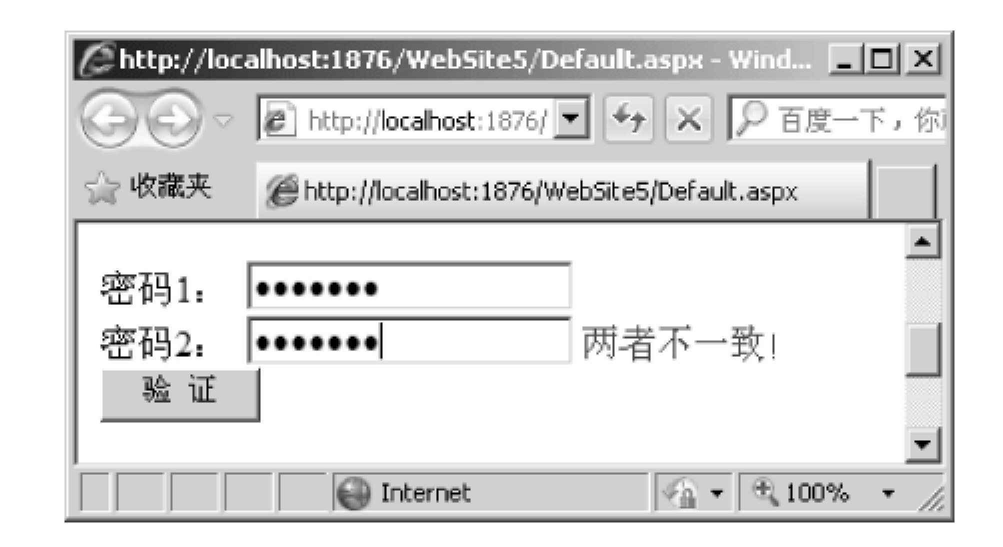


图 7-13　CompareValidator 验证控件

查用户的输入是否在指定的最大值与最小值之间。通常情况下用于检查数字、日期、货币等。其常用属性如表 7-15 所示。

表 7-15　范围验证控件的属性

属　　性	说　　明
ControlToValidate	要验证的控件 ID
MaximumValue	指定有效范围的最大值
MinimumValue	指定有效范围的最小值
Type	要比较的值的数据类型，取值有 String、Integer、Date、Double、Currency
ErrorMessage	出错提示信息

RangeValidator 控件的示例代码如下。

```
<form id = "form3" runat = "server">
  <div>请输入生日:
    <asp:TextBox ID = "TextBox1" runat = "server"></asp:TextBox>
    <asp:RangeValidator ID = "RangeValidator1" runat = "server"
        ControlToValidate = "TextBox1" ErrorMessage = "超出规定范围"
        MaximumValue = "2012/1/1" MinimumValue = "1990/1/1" Type = "Date">
    </asp:RangeValidator>
    <br />
    <asp:Button ID = "Validate3" runat = "server" Text = " 验 证 " />
    </div>
</form>
```

上述代码中，要求用户输入生日的日期，MinimumValue 属性和 MaximumValue 属性分别指定了输入范围的下限和上限，比较类型为日期型。当用户输入超出范围时，则提示错误，如图 7-14 所示。

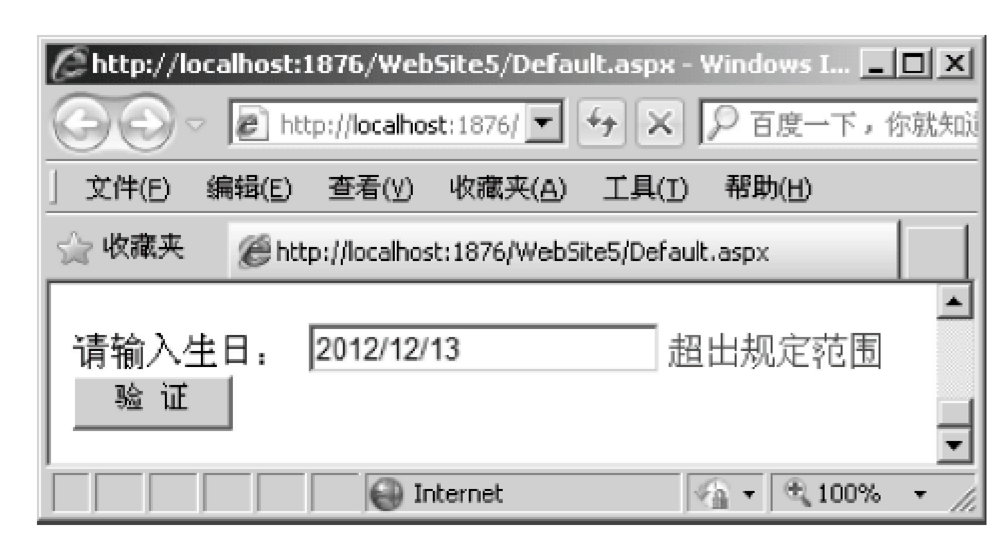


图 7-14　RangeValidator 验证控件

7.4.4 正则表达式验证控件

在实际的验证过程中，经常需要对用户输入进行一些复杂的格式验证。例如，要求用户按照"(区号)电话号码"的格式输入电话号码，或按照电子邮件、身份证号的格式进行输入等，这就需要用到正则表达式验证控件(RegularExpressionValidator)。

所谓正则表达式，就是比通常用的 * 和? 通配符更复杂的一种字符串定义规则。

例如：

```
[a～zA～Z]{3,6}[0～9]{6}
```

该正则表达式表示可以输入 3～6 个任意字母和 6 个数字。其中的限定符含义如表 7-16 所示。

表 7-16 正则表达式中的限定符

限 定 符	说 明
[]	表示可以输入的字符表达式
a～z	表示所有的小写字母
A～Z	表示所有的大写字母
0～9	表示所有的数字
{n}	表示限定的表达式必须出现 n 次
{n,}	表示限定的表达式至少出现 n 次
{n,m}	表示限定表达式必须出现 n 到 m 次
{}	表示一个字符
.{0,}	表示任意一个字符
*	表示所限定的表达式出现 0 次或多次
?	表示所限定的表达式出现 0 次或 1 次
+	表明一个或多个元素将被添加到正在检查的表达式
|	表示"或"
\	匹配限定符本身
\d	指定输入的值是一个数字
\w	表示允许输入任何值

使用正则表达式能够实现强大的字符串格式匹配验证工作，RegularExpressionValidator 控件的功能就是确定输入控件的值是否与某个正则表达式所定义的模式相匹配。在该控件的属性列表中，选择 ValidationExpression 属性，可以看到系统提供的常用正则表达式，如图 7-15 所示。

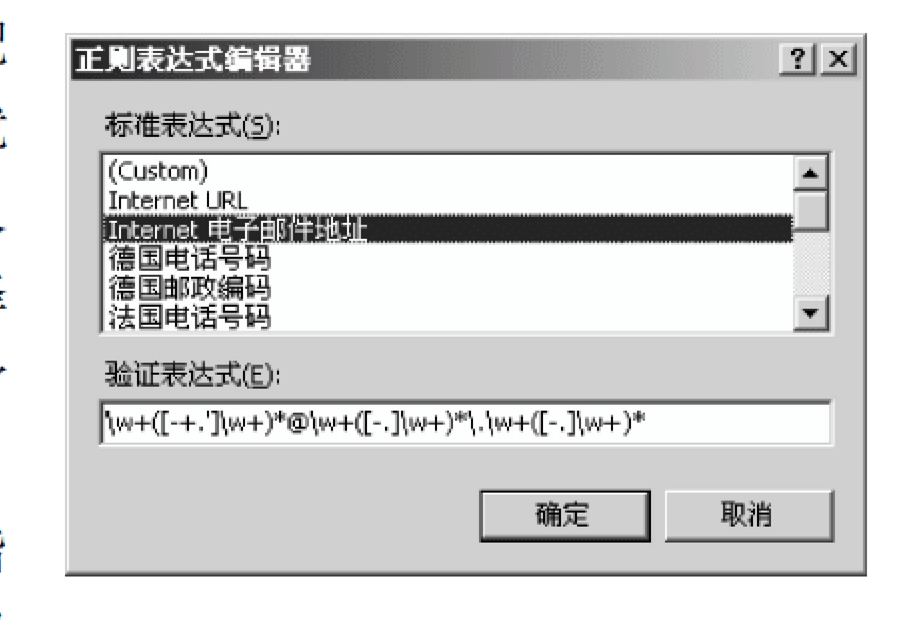


图 7-15 系统提供的正则表达式

当在系统提供的正则表达式中进行了选择，并指定了要验证的控件后，系统自动生成的 HTML 代码如下。

```
<asp:RegularExpressionValidator ID = "RegularExpressionValidator1" runat = "server"
      ControlToValidate = "TextBox1"
      ErrorMessage = "格式不匹配"
      ValidationExpression = "\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
```

上述代码中，属性 ValidationExpression 指定正则表达式，程序运行后，当用户单击按钮时，如果输入的信息与正则表达式不匹配，则提示错误信息，如图 7-16 所示。

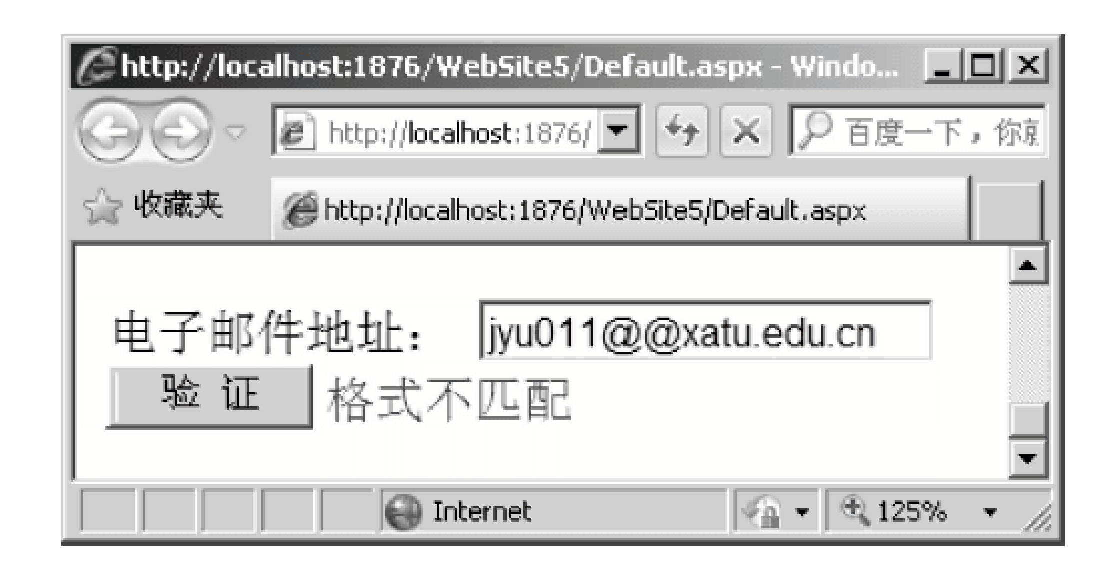


图 7-16　RegularExpressionValidator 验证控件

同样，开发人员也可以自定义正则表达式来规范用户的输入。例如，比较常用的正则表达式有：

- 只许输入数字：[0-9]*
- 只许输入 n 位的数字：\d{n}或[0-9]{n,}
- 只能输入至少 n 位的数字：\d{n,}或[0-9]{n,}
- 只能输入 n～m 位的数字：\d{n,m}
- 可以输入 3 到 6 个字母：[a-zA-Z]{3,6}
- 可能输入由 26 个字母组成的字符串：[A-Za-z]+
- 只能输入由数字和 26 个字母组成的字符串：[A-Za-z0-9]+
- 验证 E-mail 格式：.{1,}@.{1,}\.[a-zA-Z]{2,3}
- 电话号码：[0-9]{3,4}-[0-9]{7,8}
- 18 位身份证号码：
- [0-9]{6}[12][0-9]{3}[01][0-9][0123][0-9][0-9]{3}[12]

RequiredFieldValidator 控件通常与文本框控件一起使用，以检查电子邮件 ID、电话号码、信用卡号码、用户名和密码等是否有效。

需要注意的是，当用户输入为空时，除了 RequiredFieldValidator 验证控件外，其他的验证控件都会验证通过。所以，在验证控件的使用中，通常需要同 RequiredFieldValidator 控件一起使用。

7.4.5　自定义验证控件

前面讲述的数据验证控件已经提供了许多验证功能，然而有时还需要特殊的数据验证。

例如,网上购物时需要验证用户提供的银行账户中是否有足够的余额可供支付货款,等等。为了满足这种数据验证的需要,可以使用自定义验证控件(CustomValidator)。

CustomValidator 控件的使用方法与其他验证控件的使用方法基本一致,也有 ControlToValidate 和 ErrorMessage 等属性。其特殊之处在于它提供了一个 ServerValidate 事件,可以在此事件中编写完成数据验证的代码。

包含 CustomValidator 控件的 HTML 代码如下。

```
<form id="form4" runat="server">
<div>请输入系统密码:
<asp:TextBox id="passwd4" TextMode="Password" runat="server" />
<asp:CustomValidator id="CustomValidator1" runat="server"
      ControlToValidate="passwd4" ErrorMessage="输入的密码不正确"
      onservervalidate="CustomValidator1_ServerValidate">
</asp:CustomValidator>
<br />
<asp:Button id="btnLogin" runat="server" text=" 登 录" onclick="btnLogin_Click" />
<asp:Label ID="lblmessage" runat="server" Text=""></asp:Label>
</form>
```

ServerValidate 事件的处理程序如下。

```
protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs args)
{
    string strVal = args.Value.ToUpper();
    if ( strVal.Equals("ADMINISTRATOR") )
    {
        args.IsValid = true;
    }
    else
    {
        args.IsValid = false;
    }
}
protected void btnLogin_Click(object sender,System.EventArgs e)
{
    if (CustomValidator1.IsValid )
    {
        lblmessage.Text = "密码验证通过!";
    }
}
```

上述程序运行后,在文本输入框中输入 administrator,则密码输入正确,页面显示结果如图 7-17 所示。

此外,要特别注意一下当数据为空时的处理方法。默认情况下,如果相关联的输入控件为空,CustomerValidator 控件将不进行数据验证的工作。如果需要处理输入为空的情况,可将 ValidateEmptyText 属性设置为 True,此时,如果单击"登录"按钮,将显示 CustomerValidator 控件设定的出错信息,即"输入的密码不正确"。

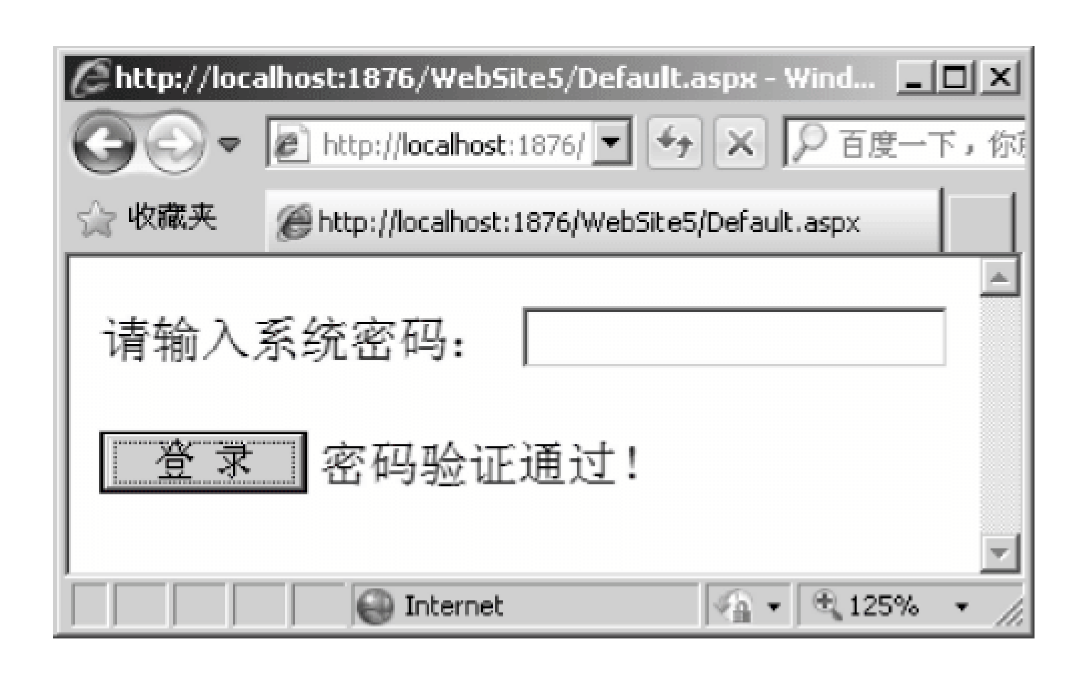


图 7-17　CustomValidator 控件

7.4.6　验证总结控件

验证总结控件(ValidationSummary)本身并不提供任何验证,但可以和前面讲过的其他验证控件一起使用,即对同一页面的多个验证控件集中给出验证结果。也就是说,当前页面有多个错误发生时,ValidationSummary 控件能够同时捕获多个验证错误并一起呈现给用户,其显示的错误信息摘要都是由该页面的其他验证控件的 ErrorMessage 属性提供的。

ValidationSummary 控件的常用属性如表 7-17 所示。

表 7-17　验证总结控件的属性

属　性	说　明
DisplayMode	错误摘要的显示方式。取值包括 List(列表)、BulletList(项目符号列表)和 SingleParagraph(单个段落)
HeaderText	在错误摘要中显示的标题文本
ShowMessageBox	是否在弹出消息框中显示错误摘要,默认为 False
ShowSummary	是否在页面上显示错误摘要,默认为 True

注意:Page. IsValid 属性检查页面中的所有验证控件是否均已成功进行验证。该属性为 Web 窗体页中的一个属性,如果页面验证成功,则将具有值 True;否则将具有值 False。

例如:

```
private void ValidateBtn_Click(Object Sender, System.EventArgs e)
{
    if (Page.IsValid == true)
    {
        lblMessage.Text = "页面有效";
    }
    else
    {
        lblMessage.Text = "页面中存在一些错误";
    }
}
```

7.5 用户控件

用户控件是一种自定义的、可复用的组合控件，通常由系统提供的可视化控件组合而成。程序员可以将一些反复使用的用户界面(既包括页面代码，也包括事件处理程序)封装成一个控件，然后可以像使用普通 Web 控件一样使用该控件。

7.5.1 用户控件概述

1. 用户控件的基本特点

要理解用户控件，需要明确如下几点：

- 用户控件是实现代码与内容分离、代码重用的技术。
- 可以像设计 Web 窗体一样设计用户控件，并定义其属性和方法。
- 用户控件可以单独编译，但不能单独运行，必须嵌入到 Web 页面中才能运行。
- 用户控件可以在第一次请求时被编译并存储在服务器内存中，从而缩短以后请求的响应时间。

2. 用户控件与 Web 页面的比较

用户控件与普通 Web 页面非常相似，都具有自己的用户界面和程序代码。创建用户控件所采用的方法与创建 Web 页面的方法基本相同。程序员可以使用任何文本编辑器创作用户控件，或者使用代码隐藏类开发用户控件。

用户控件与普通 Web 页面之间也存在一些不同：

- 用户控件的文件扩展名为 ascx 和 ascx. cs，ASP.NET 页面的文件扩展名是 aspx 和 aspx. cs。
- 用户控件不包含<html>、<body>和<form>标记。
- 用户控件不含@Page 指令，而是@Control 指令。
- 不能独立地请求用户控件，用户控件必须包括在 Web 窗体页内才能使用。

7.5.2 创建用户控件

用户控件是封装成可复用控件的 Web 窗体，可以使用标准 Web 窗体页面上相同的 HTML 元素和 Web 控件来设计用户控件。要创建一个用户控件，可以有两种方式：一种是直接创建用户控件；另一种是将已经设计完成的 Web 窗体页面改为用户控件。

1. 直接创建用户控件

创建步骤如下：

(1) 添加一个新的用户控件。

打开解决方案资源管理器，右击项目名称，选择“添加新项”，则出现如图 7-18 所示的添加新项对话框，选择“Web 用户控件”，默认的用户控件名称为 WebUserControl. ascx，假设修改名称为 myControl. ascx，然后单击“添加”按钮，则该用户控件会添加到解决方案资源管理器的项目列表中。



图 7-18　添加新项对话框

此时，可以看到该用户控件默认包含了一行代码：

```
<% @Control  Language = "C#" AutoEventWireup = "true"
CodeFile = "myControl.ascx.cs"
Inherits = "myControl"  %>
```

说明：

① @Control 指令用来标识用户控件，正如@page 指令用来标识 Web 窗体一样。

② Language="C#" 用于指定用户控件的编程语言是 C#。

③ AutoEventWireup="true" 指示控件的事件与处理程序可以自动匹配。

④ CodeFile="myControl.ascx.cs" 指定所引用的控件代码文件的路径。

⑤ Inherits="myControl" 指定用户控件是从 myControl 类派生的。

(2) 向用户控件的设计页面(myControl.ascx)中添加各种 Web 控件并修改其属性。

例如，创建一个用户控件 myControl.ascx，如图 7-19 所示。其中包括了两个 Web 控件：Textbox 控件和 Button 控件。

上述用户控件的页面代码如下：

```
<% @Control Language = "C#" AutoEventWireup = "true" CodeFile = "myControl.ascx.cs" Inherits
 = "myControl"  %>
<p> 请输入验证码
<asp:TextBox ID = "TextBox1" runat = "server"></asp:TextBox>
</p>
<p><asp:Button ID = "Button1" runat = "server" Text = "确定" /></p>
```

(3) 在代码文件(myControl.ascx.cs)窗口中，给该用户控件的子控件编写事件响应代码。

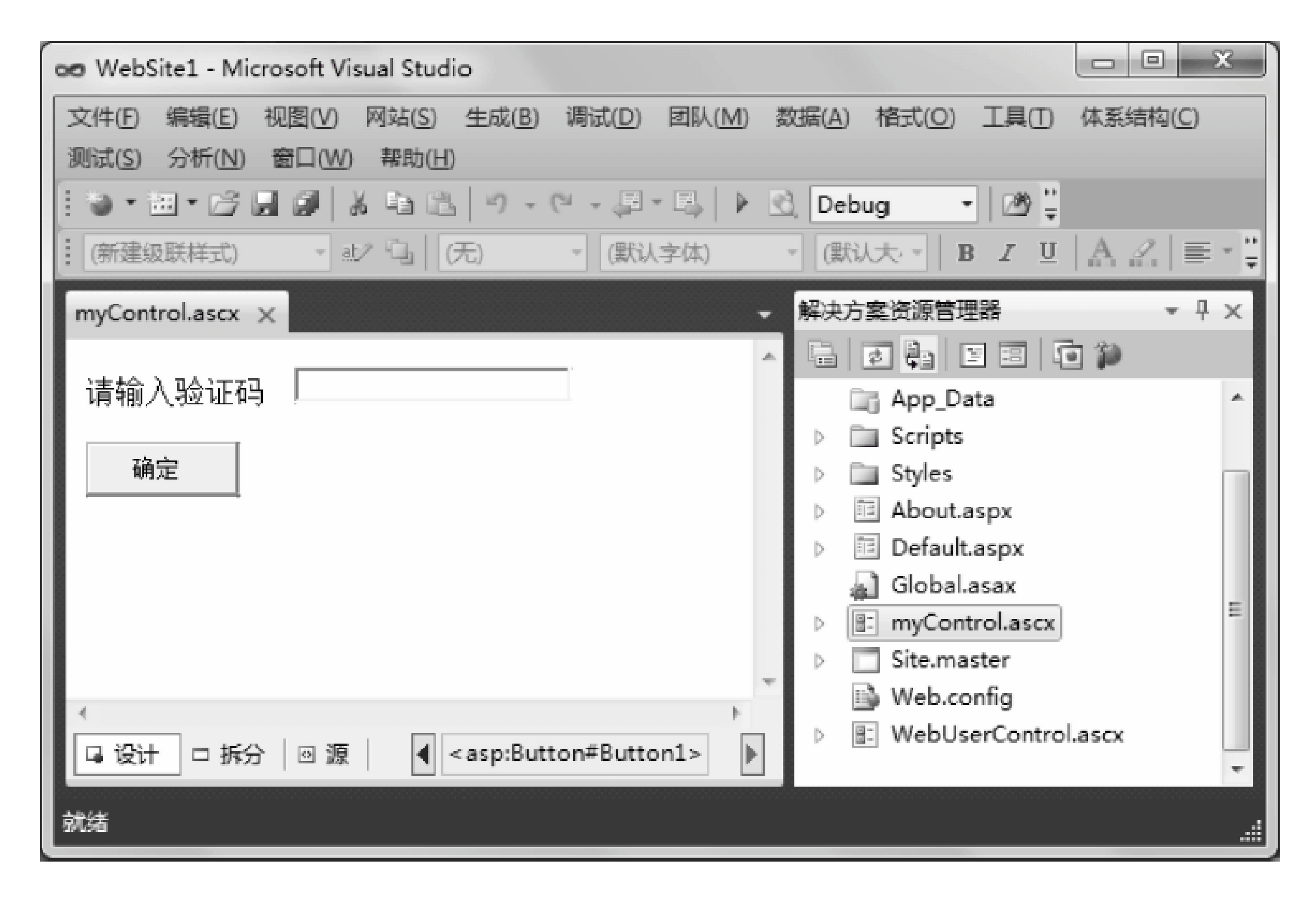


图 7-19 用户控件示例

在 myControl.ascx.cs 文件的编辑窗口，可以编写所有子控件的事件处理代码，如按钮单击事件 Button1_Click 等。

```
protected void Button1_Click(object sender, EventArgs e)
{
    …
}
```

至此，一个用户控件就创建完成了。

2. 将已有的 Web 窗体页面改为用户控件

步骤如下：

(1) 删除 aspx 文件中的<HTML>、<BODY>和<FORM>等标记，因为它们可能与包含页面有冲突。

(2) 将@page 指令改为@control 指令。

(3) 在代码文件 cs 中定义的类的基类由 Page 类改为 UserControl 类。

(4) 将文件的扩展名改为 ascx

值得注意的是，由于用户控件不能作为一个独立的网页来显示，创建了用户控件后，必须添加到其他 Web 窗体页面中才能显示运行。因此，用户控件不能设置为“初始页面”。

7.5.3 用户控件的使用

将用户控件添加到 Web 窗体页面中，也有两种方法。一种是向 Web 窗体页面添加该控件的@Register 指令和标记，使得用户控件成为页面的一部分，这样就能够在页面显示和使用；另一种方法是通过编程方式向页面动态地添加用户控件。

下面分别介绍这两种使用方式。

(1) 直接在 Web 窗体页面添加用户控件。其步骤如下：

① 在页面的顶部添加一个注册该控件的@Register 指令，以便在处理 Web 窗体页面时

识别该控件。其格式如下：

```
<%@ Register
    TagPrefix = "namesapce"
    TagName = "controlname"
    Src = "controlpath"  %>
```

说明：

- TagPrefix 表示确定控件的唯一命名空间，是标记中控件名称的前缀。
- TagName 表示控件的名称，即标识该控件的一个标记。
- Src 表示用户控件的路径。

② 在页面的主体部分，使用标记的形式显示该控件。即使用 TagPrefix 和 TagName 形成一个标记，并为该控件指定一个 ID 和设置 runat 属性。其形式如下：

```
<Tagprefix:Tagname  id = "ControlID"  runat = "server" />
```

例如，要在 Web 页面中使用在 7.5.2 节中创建的用户控件 myControl.ascx，则需要在 aspx 文件中编写如下代码：

```
<%@Register TagPrefix = "myPre" TagName = "myName" Src = "myControl.ascx"  %>
…
<myPre:myName id = "myCon" runat = "server" />
```

一个页面可以放置同一种用户控件的多个实例，只需要保证其 id 值不同即可。

(2) 通过编程方式动态地添加用户控件。其步骤如下：

① 使用@Reference 指令注册该用户控件。

② 使用 LoadControl 方法添加该控件。

③ 最后使用 Page 类的 Controls.Add()方法将该控件加载到该页面。

例如：

```
<%@ Reference control = "myControl.ascx"  %>
Control c1 = LoadControl("myControl.ascx");
Page.Controls.Add(c1);
```

7.6 习题与上机练习

1. 选择题

(1) 下列控件中，不能执行单击事件的是(　　)。

A. ImageButton　　B. ImageMap　　C. Image　　D. LinkButton

(2) 下面不属于容器控件的是(　　)。

A. Panel　　B. CheckBox　　C. Table　　D. PlaceHolder

(3) 下面对 ASP.NET 验证控件说法正确的是(　　)。

A. 可以在客户端直接验证用户的输入信息并显示错误信息

B. 对一个下拉表控件，不能使用验证控件

C. 服务器验证控件在执行验证时，必定在服务器端执行

D. 对验证控件，不能自定义规则

(4) 要将 Textbox 控件设置成多行输入，TextMode 属性须设置成(　　)。

A. Singleline　　B. Multiline　　C. Password　　D. Textarea

(5) 下面的(　　)控件不能对 Web 页上的输入控件进行验证。

A. RangeValidator　　B. ValidationSummary

C. RegularExpressionValidator　　D. CompareValidator

(6) 下面对 CustomValidator 控件说法错误的是(　　)。

A. 能自定义的验证函数

B. 可以同时添加客户端验证函数和服务器端验证函数

C. 指定客户端验证的属性是 clientValidationFunction

D. 属性 runat 用来指定服务器端验证函数

(7) 如果需要确保用户输入大于 100 的值，应该使用(　　)验证控件。

A. RequiredFieldValidator　　B. RangeValidator

C. CompareValidator　　D. RegularExpressionValidator

2. 填空题

(1) 所有 Web 服务器控件都位于________名称空间，而且大多数 Web 服务器控件都是从基类________直接或间接派生的。

(2) CheckBox 控件的________属性值指示是否已选中该控件。

(3) 使用 ListBox 控件的________属性获取列表控件项的集合，使用________属性获取或设置该控件的选择模式。

(4) 对于 DropDownList 控件，使用________属性获取列表控件中选定项的值；当列表控件的选定项在信息发往服务器之间变化时发生________事件。

(5) 设置属性________可决定 Web 服务器控件是否可用。

3. 简答题

(1) 如何判断页面是第一次被加载执行？

(2) 说明<a>元素、LinkButton 和 HyperLink 控件的区别。

(3) Button、LinkButton 和 ImageButton 控件有何不同？单击这些控件时会发生 Click 事件和 Command 事件，这两个事件有何区别？

4. 上机练习

(1) 创建一个 Web 窗体页，并添加一个 Image 控件和一个 Button 控件，要求单击按钮时更改 Image 控件显示的图片。

(2) 编写用户注册页面 register.aspx，选择适当的 Web 服务器控件实现如下功能：输入姓名、性别、生日、出生地、联系电话、地址等，必要时用验证控件进行验证。当用户按下“提交”按钮后显示输入的信息。

(3) 编写程序 exam.aspx，显示 5 个单项选择题，用户选择答案并提交后给出分数。

第 8 章　ASP.NET 的对象

Web 应用程序在传统意义上来说是无状态的，Web 应用不能像 Win Form 那样维持客户端状态，所以在 Web 应用中，通常需要使用内置对象进行客户端状态的保存。这些内置对象能够为 Web 应用程序的开发提供设置、配置以及检索等功能。

.NET Framework 包含一个内置的对象类库。在脚本中，可以不必创建这些对象的实例而直接访问它们的属性、方法和数据集合。通过这些对象，可以实现获取客户端请求、输出响应信息、应用程序会话管理、存储用户信息、保存状态信息等功能，如表 8-1 所示。

表 8-1　ASP.NET 内置对象

对象名	功　能	ASP.NET 类
Page	用于设置与网页有关的属性、方法和事件	
Request	读取客户端所提交的数据	HttpResponse
Response	发送信息到客户端	HttpRequest
Application	为所有用户提供共享信息	HttpApplicationState
Session	存储特定用户的信息，可以在同一网站的多个页面间共享信息	HttpSessionState
Cookie	在客户端的磁盘上保存用户的数据	HttpCookie
Server	提供服务器端的属性和方法	HttpServerUtility
Context	封装了每个用户的会话、当前 HTTP 请求和请求页的信息	HttpContext
ViewState	在同一个页面的多次请求间保存状态信息	StateBag
Trace	用于对页面进行跟踪	TraceContext

8.1　HTTP 请求处理

8.1.1　Response 对象

Response 对象属于 HttpResponse 类，主要用于动态响应客户端的请求，包括直接发送信息给客户端、重定向 URL、在客户端设置 Cookies 等。Response 对象将服务器端动态生成的结果以 HTML 格式返回到客户端的浏览器。

1. Response 对象的方法

Response 对象的常用方法如表 8-2 所示。

表 8-2 Response 对象的方法

方 法	说 明
AppendCookie	将一个 HTTP Cookie 添加到内部 Cookies 集合
AppendToLog	将日志信息添加到 IIS 日志文件中
BinaryWrite	将一个二进制字符串写入 HTTP 输出流
Clear	清除服务器缓存中的所有数据
Flush	把服务器缓存中的数据立刻发送到客户端
End	停止处理当前文件,并返回结果
Redirect	用于页面的跳转,使浏览器重定向到另一个 URL
Write	直接向客户端输出数据
WriteFile	向客户端输出文本文件的内容

2. Response 对象的属性

Response 对象的常用属性如表 8-3 所示。

表 8-3 Response 对象的属性

属 性	说 明
Buffer	指定页面输出时是否需要缓冲区
Cache	获得网页的缓存策略(过期时间、保密性等)
Charset	设置输出到客户端的 HTTP 字符集
ContentEncoding	获取或设置输出流的 HTTP 字符集
ContentType	获取或设置输出流的 MIME 类型,默认为 text/HTML
Cookies	用于获得 HttpResponse 对象的 Cookie 集合
Expires	设置页面在浏览器中缓存的时限,以分钟为单位
IsClientConnected	表明客户端是否与服务器端连接,其值为 True 或 False
Output	启用到输出 HTTP 响应流的文本输出
Status	服务器返回的状态行的值

下面通过几个简单的示例说明 Response 对象的属性和方法的应用。

例 8-1 检查客户端的联机状态(使用 IsClientConnected 属性和 Redirect 方法)。

```
protected void Page_Load(object sender, EventArgs e)
{
    //判断客户端是否与服务器连接
    if  (Response.IsClientConnected)
        Response.Redirect("other.aspx");     //跳转到当前目录的另一个页面
    else
        Response.End();                      //停止执行当前页面的代码
}
```

上述代码中,通过 IsClientConnected 属性指示客户端是否仍连接在服务器上,如果是,则实现页面跳转,否则停止当前页面的执行。

例 8-2 判断向客户端的输出(使用 Buffer 属性和 Flush、Clear、Write 方法等)。

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
    //设置服务器缓冲为 true
    Response.Buffer = true;
    //获取当前时间的小时数
    int currentHour = DateTime.Now.Hour;
    //满足某个特定条件
    if (currentHour == 0)
    {
        Response.Write("时间到,服务器将停止输出");
        Response.Flush();         //立即发送缓冲区中的数据
        Response.Clear();         //清除缓冲区中的全部数据
        Response.End();           //停止执行代码
    }
    else
    {
        Response.Write("服务器正常工作中…");
    }
}
```

在实际应用中,服务器可能需要取消向客户端的输出,这时可以先通过 Clear 方法清除缓冲区,然后利用 End 方法停止输出操作。

3. 使用 Response 对象设置 Cookie

Response 对象的数据集合只有一个,那就是 Cookies 集合。利用 Response.Cookies 的 Add 方法可以创建一个新的会话 Cookie,格式如下:

```
Response.Cookies.Add(Cookies 对象名);
```

例如:

```
//创建一个 Cookie
HttpCookie myCookie = new HttpCookie("Username");
myCookie.Value = "张三";
//将新 Cookie 加入 Cookies 集合中
Reponse.Cookies.Add(myCookie);
```

上述代码也可以简化为:

```
Reponse.Cookies["Username"] = "张三";
```

8.1.2 Request 对象

Request 对象用于获取从客户端的浏览器提交给服务器的信息。这些信息包括 HTML 表单数据、URL 地址后的附加字符串、客户端的 Cookies 信息、用户认证等。

1. Request 对象的属性

Request 对象的常用属性如表 8-4 所示,包括 4 种常见的数据集合(Collection):QueryString 集合、Form 集合、Cookies 集合和 ServerVariables 集合。

表 8-4　Request 对象的常用属性

属　性	说　明
Browser	取得客户端浏览器的信息
ClientCertificate	取得当前请求的客户端安全证书
ContentEncoding	取得客户端浏览器的字符设置
ContentType	取得当前请求的 MIME 类型
Cookies	数据集合,取得客户端发送的 Cookies 数据
HttpMethod	当前客户端提交数据的方式(Get/Post)
Form	数据集合,取得客户端利用 POST 方法传递的数据(页面中定义的窗体变量的集合)
QueryString	数据集合,取得客户端利用 GET 方法传递的数据(以"名/值"对表示的 HTTP 查询字符串变量的集合)
Params	获得以名/值对表示的 QueryString、Form、Cookie 和 ServerVariables 组成的集合
ServerVariables	数据集合,取得 Web 服务器端的环境变量信息
TotalBytes	从客户端接收的所有数据的字节大小
UserHostAddress	取得客户端主机的 IP 地址
UserHostName	取得客户端主机的 DNS 名称

引用集合的格式为:

```
Request. 集合名 ("变量名")
```

下面根据功能分别对这些集合加以介绍。

1) 要获取客户端提交的表单数据,则需用到 QueryString 集合和 Form 集合。

客户端通过 Form 表单向服务器提交数据需有 POST 和 GET 两种方法。

① 当客户端使用 GET 方法提交时,服务器就通过 QueryString 集合获取数据。GET 方法将表单数据作为参数直接附加到 URL 地址的后面,附加参数和 URL 地址之间通常用"?"连接。由于浏览器对地址栏的长度有限制,因此也限定了提交数据的长度。

附加参数的格式为:

```
URL?Variable = Value
URL?Variable1 = Value1 & Variable2 = Value2
```

其中,Varibale 是通过 HTTP 传递过来的变量名或 GET 方式提交的表单变量,Value 是变量的值。当有多个参数变量时,则不同"名/值"对之间用"&"符号连接。

例如:

```
http://www.sina.com/news.asp?userid = 22306    //符号?后的 userid = 22306 是一个"名/值"对
```

服务器获取数据时采用

```
Request.Querystring ("userid")                 //读取表单中 userid 输入域的值
```

当客户端有大量信息需提交时，通常使用 POST 方法。这样服务器就通过 Form 集合取出用户输入的数据。

例如：

```
Request.Form ("userid")
```

注意：可以省略 Request 后的 Querystring 或 Form 集合名，而直接采用 Request(“变量名”)的形式。例如，直接使用 Request(“username”)，等价于 Request. Form(“username”)或 Request. Querystring (“username”)的结果。

2）读取保存的 Cookie 信息

要从 Cookies 中读取数据，则要使用 Request 对象的 Cookies 集合，其格式为：

```
Request.Cookies["Cookies 名称"]
```

例如：

```
//获取 Cookie 对象
HttpCookie MyCookie = Request.Cookies["Username"];
//读取 Cookie 值
string myName = MyCookies.Value;
```

在 BBS 或聊天室中，常常将用户登录时输入的用户名或昵称(如 nickname)保存在 Cookie 中，这样在后面的程序中就可以容易地调用该用户的昵称了。

3）读取服务器端的环境变量

ServerVariables 集合用于获取系统的环境变量信息。使用的语法格式为：

```
Request.ServerVariables("环境变量名")
```

或

```
Request("环境变量名")
```

例如，使用下面的语句能够在页面中显示客户端的 IP 地址：

```
Request. ServerVariables("REMOTE_ADDR")
```

表 8-5 所示是部分的环境变量名，可以得到 ServerVariables 集合内存储的对应的变量值。

表 8-5　ServerVariables 环境变量

环境变量名	说　　明
ALL_HTTP	客户端发送的所有 HTTP 标题
CERT_COOKIE	客户端验证的唯一 ID，以字符串方式返回
CONTENT_LENGTH	客户端提交的正文长度
CONTENT_TYPE	正文的数据类型。可用于判断用户提交数据的方法，如 GET、POST 等
HTTP_ACCEPT_LANGUAGE	获取客户端所使用的语言

续表

环境变量名	说　　明
LOCAL_ADDR	返回接受请求的服务器 IP 地址
PATH_INFO	获取虚拟路径信息
PATH_TRANSLATED	获取当前页面的物理路径
QUERY_ STRING	查询 HTTP 请求中问号(?)后的信息
REMOTE_ADDR	发出请求的远程主机的 IP 地址
REMOTE_HOST	发出请求的主机名称
REMOTE_USER	用户发送的未映射的用户名字符串。该名称是用户实际发送的名称,与服务器上验证过滤器修改过后的名称对应
REQUEST_METHOD	获取表单提交内容的方法,如 GET、POST 等
SCRIPT_NAME	执行脚本的虚拟路径或自指定的 URL 路径
SERVER_NAME	获取服务器主机名、DNS 别名、IP 地址以及自指定的 URL 路径
SERVER_PORT	发出请求的端口号
SERVER_PROTOCOL	请求信息协议的名称和修订版本,格式为 protocol/revision
SERVER_SOFTWARE	服务器运行的软件名称和版本号,格式为 name/version
URL	系统的 URL 路径

2. Request 对象的方法

Request 对象的常用方法如表 8-6 所示。

表 8-6　Request 对象的方法

方　　法	说　　明
BinaryRead	执行对当前输入流进行指定字节数的二进制读取
MapPath	将请求 URL 中的虚拟路径映射到服务器上的物理路径
GetType	获取当前对象的类型
SaveAs	将 HTTP 请求保存到磁盘
ToString	将当前对象转换成字符串

一般来说,使用 BinaryRead 方法取得服务器端所传递的数据就不能使用 Request 对象所提供的各种数据集合,否则会发生错误。反之,若使用 Request 对象的数据集合取得客户端数据,也不能使用 BinaryRead 方法。因此,BinaryRead 方法并不常用。

8.1.3　Server 对象

Server 对象属于 HttpServerUtility 类。该类包含处理 HTTP 请求的方法。

通过 Server 对象可以访问服务器上的方法和属性。例如,得到服务器上某文件的物理路径、设置某文件的执行期限等。使用 Server 对象可以创建各种服务器组件实例,从而实现服务器上的许多高级功能,如访问数据库、文件输入输出等操作。

1. Server 对象的属性

HttpServerUtility 类是 Server 对象对应的 ASP.NET 类。它的属性如表 8-7 所示。

Server 对象的属性 ScriptTimeout 用来表示 Web 服务器响应一个网页请求所需要的时间。如果脚本超过该时间限度还没有执行结束,它将被强行中止,并提交超时错误。这样做主要是用来防止某些可能进入死循环的错误导致服务器过载问题。

表 8-7　Server 对象的属性

属　　性	说　　明
ScriptTimeout	获取和设置脚本文件执行的最长时间(单位为秒)
MachineName	获取服务器的计算机名称

如果不对 ScriptTimeout 属性进行设置,则默认为 90 秒。如果将其设置为－1,则永远不会超时。该语句要放在所有 ASP 执行语句之前,否则不起作用。

例如:

```
Server.ScriptTimeout = 100      //指定服务器处理脚本的超时期限为 100 秒
limit = Server.ScriptTimeout    //将设置过的超时期限存放到一个变量中
```

2. Server 对象的方法

Server 对象的方法及含义如表 8-8 所示。

表 8-8　Server 对象的方法

方　　法	说　　明
ClearError	清除前一个异常
CreateObject	创建 COM 对象的一个服务器实例
Execute	停止执行当前网页,转到另一个网页执行,执行完后返回原网页
GetLastError	获得前一个异常,可用于访问错误信息
Transfer	停止执行当前网页,转到新的网页执行,执行完后不返回原网页
MapPath	将指定的虚拟路径映射为物理路径
HTMLEncode	对字符串进行 HTML 编码,并将结果输出
HTMLDecode	对 HTML 编码的字符串进行解码,并将结果输出
UrlEncode	将字符串按 URL 编码规则输出
UrlDecode	对在 URL 中接收的 HTML 编码字符串进行解码,并将结果输出

1) CreateObject 方法

CreateObject 方法用于在 ASP.NET 中创建一个服务器实例。

例如:

```
Server.CreateObject ("ADODB.Connection")
```

表示创建一个 ADO 组件的实例。

2) Execute 方法和 Transfer 方法

Execute 方法调用一个指定的脚本文件并且执行,执行完毕后再返回原文件。就像被调用的脚本文件存在于这个主文件中一样,类似于许多语言中的类或子程序的调用。

Execute 方法的语法格式为:

```
Server.Execute(string URL)
```

其中,参数 URL 为指定执行的脚本文件的地址。

Transfer 方法的作用是将一个正在执行的脚本文件的控制权转移给另一个文件执行,

执行完毕后不返回原文件。其语法格式为：

```
Server.Transfer(string URL)
```

注意：Server 对象的 Transfer 方法、Execute 方法和 Response 对象的 Redirect 方法既有相似之处，也有区别。它们都是终止执行当前 Web 页面，转而执行另一个页面。

- 使用 Transfer 和 Execute 方法只能转移到本网站的其他网页，而 Redirect 方法可以转移至任一网站的任一个网页。
- Execute 方法相当于子程序的调用，执行被调用程序后，会返回原程序。Transfer 方法、Redirect 方法都不再返回原程序执行。
- 使用 Transfer 方法调用另一个文件，会进行控制权的转移，并且所有内置对象的值都一起"转移"，并保留至新的网页。Redirect 方法则仅仅转移控制权。
- 使用 Transfer 和 Execute 方法时，客户端与服务器只进行一次通信。而 Redirect 方法在重定向过程中，客户端与服务器进行两次来回的通信。第一次通信是对原始页面的请求，得到一个目标已经改变的应答，第二次通信是请求指向新页面，得到重定向后的页面。

3）MapPath 方法

MapPath 方法将指定的虚拟路径映射到服务器上的物理路径。其引用格式为：

```
Server.MapPath(path)
```

其中，参数 path 是 Web 服务器上的虚拟路径，返回值是与 path 对应的物理文件路径。在虚拟路径中，以字符"/"或"\"开始的字符串，说明它是一个完整的路径，将返回一个相对于服务器根目录的地址。如果只有字符"/"或"\"，将返回服务器的根目录地址。

例 8-3 Server 对象的 MapPath 方法。

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("服务器的根路径: " + Server.MapPath("./"));
    Response.Write("<br>当前文件所在的物理路径: " + Server.MapPath(".") );
    Response.Write("<br>Default.aspx 文件的位置: " +
    Server.MapPath("Default.aspx"));
}
```

上述代码运行结果如图 8-1 所示。



图 8-1 Server 对象的 MapPath 方法

4）对 HTML 进行编码和解码

HTML 是用标记“<”和“>”括起来的，通常这些标记被浏览器识别为系统标记，不会显示在浏览器上。利用 Server 对象的 HtmlEncode 和 HtmlDecode 方法可以对 HTML 进行编码和解码。

HtmlEncode 方法对要在浏览器中显示的字符串进行编码，可以阻止浏览器解释 HTML 语法，从而直接将“<”和“>”显示在浏览器上。实际上，也就是将“<”和“>”转义为“<”和“>”发送到浏览器。

HtmlEncode 方法的引用格式为：

```
Server.HtmlEncode (string s)
Server.HtmlEncode (string s, TextWriter output)
```

其中，s 是要编码的字符串；output 是 TextWriter 输出流，包含已编码的字符串。

HtmlDecode 方法用于对已经进行 HTML 编码的字符串进行解码，是 HtmlEncode 的反操作。其语法定义如下：

```
Server.HtmlDecode (string s)
Server.HtmlDecode (string s, TextWriter output)
```

其中，s 是要解码的字符串；output 是 TextWriter 输出流，包含已解码的字符串。

例 8-4 Server 对象的 HtmlEncode 和 HtmlDecode 方法。

```
protected void Page_Load(object sender, EventArgs e)
{
    string enStr = Server.HtmlEncode("<font size=4>输出 HTML 标记</font>");
    Response.Write(enStr);
    Response.Write("<hr>");
    string deStr = "&lt;font size=5&gt;输出 HTML 标记 &lt;/font&gt;";
    Response.Write("<br>要解码的字符串: " + deStr) ;
    Response.Write("<br>解码后: " + Server.HtmlDecode(deStr));
}
```

运行结果如图 8-2 所示。

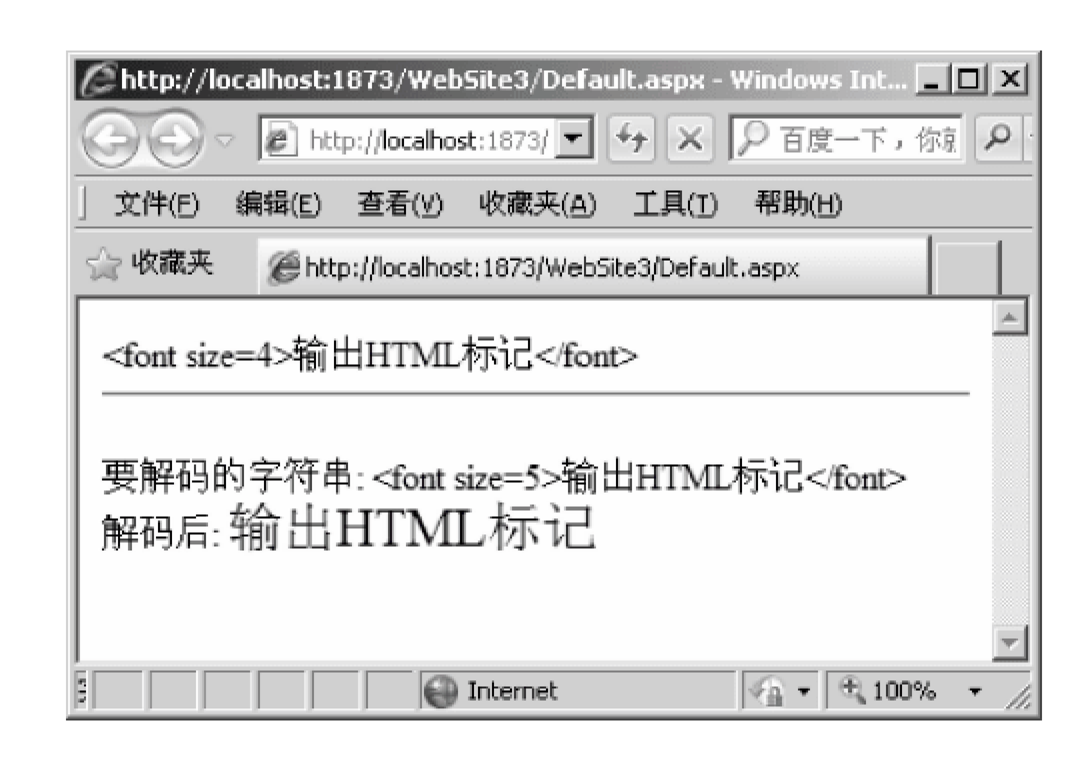


图 8-2　Server 对象的 HtmlEncode 和 HtmlDecode 方法

5）对 URL 进行编码和解码

UrlEncode 方法用于编码字符串，以便通过 URL 从 Web 服务器到客户端进行可靠的 HTTP 传输。其引用格式为：

```
Server.UrlEncode (string URL)
Server.UrlEncode (string URL, TextWriter output)
```

其中，参数 url 为要转换的 URL 地址的字符串，output 是 TextWriter 输出流，包含已编码的字符串。

在程序中，有些字符是不能被直接读取的，如空格、特殊的 ASCII 字符等。当字符串以 URL 的形式进行传递时，通常不允许出现这些字符，而根据 URL 规则对字符串进行编码后则可以传输各种字符。UrlEncode 方法将这些 ASCII 字符转换成 URL 中等效的字符。空格用"＋"代替，ASCII 码大于 126 的字符用"％"后跟 16 进制代码进行替换。

例 8-5　Server 对象的 URLEncode 方法应用。

```
protected void Page_Load(object sender, EventArgs e)
{
    String str = "http://mail.163.com" ;
    Response.Write("<p>执行 URLEncode 方法前: " + str ) ;
    String strNew = Server.URLEncode(str);
    Response.Write ("<p>执行 URLEncode 方法后: " + strNew );
}
```

运行结果如图 8-3 所示。

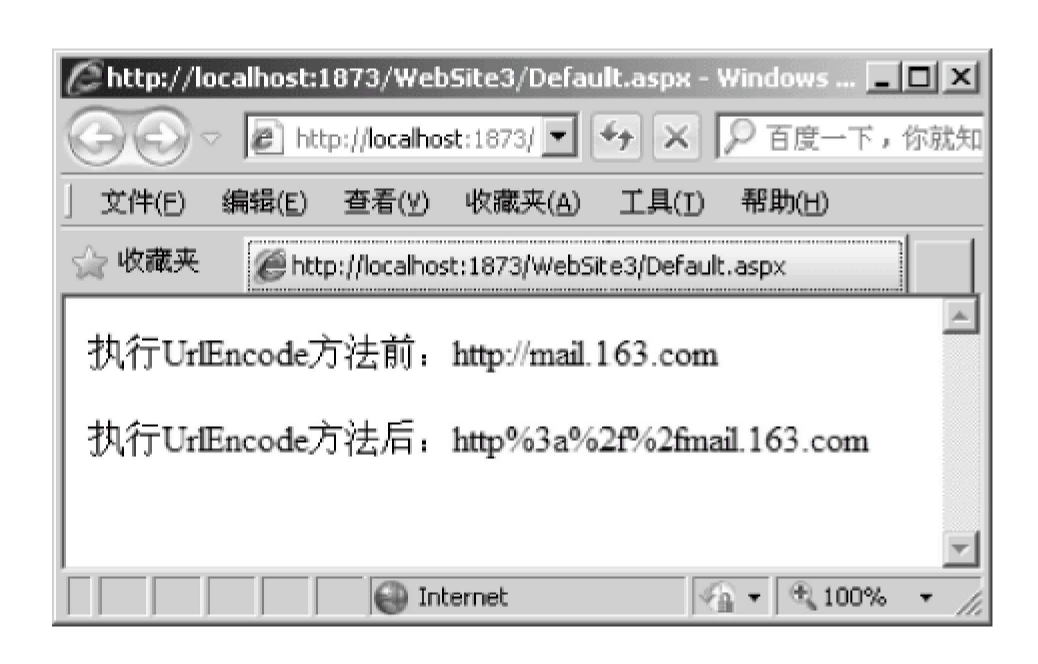


图 8-3　Server 对象的 URLEncode 方法应用

与 UrlEncode 方法相反，UrlDecode 方法用于对字符串进行解码，可以还原被编码的字符串。其引用格式为：

```
Server.UrlDecode (string URL)
Server.UrlDecode (string URL, TextWriter output)
```

8.2　状态信息保存

用户访问 Web 站点时，数据是遵从 HTTP 协议进行传输的。HTTP 是一种无状态协议，服务器对来自客户端的每个请求都视为新请求。也就是说，用户向 Web 服务器发出的

每个请求都与前面的请求无关,服务器无法知道两个连续的请求是否来自同一用户。一个典型的实例就是网上购物。假定用户访问一个网上商城,当他每次选中商品放入自己的购物车时,就向服务器发出一次单独的 HTTP 请求,如果他连续选购并放入购物车,服务器就必须记住先前放入的书(即前面的 HTTP 请求),直到用户结账或取消购物。当用户付款时,需要提供网上银行的账号和密码。此后,商家通过物流系统进行配送。用户在收到商品之前,可以随时查看自己的订单所处的状态。

在一个复杂的流程中,有大量的信息需要在各个环节进行保存、更新或查询,因此,任何一个 Web 应用系统都要解决状态信息的保存和共享问题。

下面介绍与 Web 状态信息保存有关的 Application 对象、Session 对象、Cookie 对象和 ViewState 对象等。

8.2.1 Application 对象

当需要在整个 ASP.NET 应用程序范围内共享信息时,可以使用 Application 对象。它的用途是记录整个网站的信息。也就是说,Application 对象所存储的数据可以被访问当前 Web 站点的所有用户使用,并且在网站服务器运行期间永久保存。

Application 对象是 HttpApplicationState 类的实例,存放的是供 ASP.NET 应用程序使用的变量,属于此应用程序的所有页面都可以存储并修改同一个 Application 对象(如聊天室和网站计数器)。Application 对象没有生命周期,不论客户端浏览器是否关闭,Application 对象仍然存在于服务器上。

1. Application 对象的键值

Application 对象通过使用用户自定义的数据键值来存取信息。其格式如下:

```
Application["键名"] = 值
```

例如,以下代码为 Application 对象添加一个整数。

```
Application["MyVar"] = 2;
```

在页面中,可以通过 Application["MyVar"] 读取这一数据。一旦给 Application 对象分配数据之后,它就会持久地存在,并始终占用内存空间,直到关闭 Web 服务器使得 Application 停止。

如果要删除 Application 对象中的键值,则调用 Remove() 方法,并指定键名。

例如:

```
Application.Remove["MyVar"];
```

2. Application 对象的方法

Application 对象的主要方法如表 8-9 所示。

使用 Application 对象的 Add() 方法可以向应用程序状态添加新的项。

例如:

```
Application.Add("Title", article board) 或 Application("Title") = "Article Board"
```

表 8-9　Application 对象的主要方法

方　　法	说　　明
Add()	向 Application 状态添加新对象
Clear()	从 Application 状态中移除所有对象
Remove()	从 Application 集合中按照键名移除项
Lock()	锁定 Application 对象
UnLock()	解除对 Application 对象的锁定

由于存储在 Application 对象中的数据可以并发访问，可能造成同一个变量在同一时刻被多个用户写入的情况。因而在存取 Application 对象的值之前，必须先锁定它；然后在使用完后解锁。Application 对象提供了 Lock 方法和 Unlock 方法，它们必须配对使用。

Lock 方法用于锁定 Application 对象，以确保同一时刻仅有一个用户可以修改或存取其数据键值。Unlock 方法用于解除锁定，允许其他用户修改和存取数据键值。

例 8-6　使用 Application 对象统计所有访问网站的人次(网站访问计数器)。

```
protected void Page_Load(object sender, EventArgs e)
{
    if ( Convert.ToInt32(Application["NumVisits"]) < 1)
    {
        Application["NumVisits"] = 1;
    }
    //加锁,确保同一时刻仅有一个用户可修改变量
    Application.Lock() ;
    //计数器加 1
    Application["NumVisits"] = int.Parse(Application["NumVisits"].ToString()) + 1;
    //解除锁定,允许其他用户修改计数器的值
    Application.UnLock();

    Response.Write("本网站的访问次数为 " + Application["NumVisits"].ToString() + "次!");
}
```

程序的运行结果如图 8-4 所示。



图 8-4　用 Application 对象统计访问网站的总人次

3. Application 对象的事件

Application 对象有 Application_OnStart 事件和 Application_OnEnd 事件。它们的代码放在 Global.asax 文件中。

Application_OnStart 事件在创建与服务器的首次会话(即 Session_OnStart 事件)之前发生。当服务器启动并允许用户请求时,就触发 Application_OnStart 事件。

Application_OnEnd 事件在整个 ASP.NET 应用程序退出时发生,一般用于回收占用的服务器资源,即释放 Application 变量。

表 8-10 列出了 Application 对象和 Session 对象的所有事件。

表 8-10　Application 对象和 Session 对象的事件

事　　件	说　　明
Application_Start	调用当前应用程序目录(或其子目录)下的第一个 ASP.NET 页面时触发
Application_End	应用程序的最后一个会话结束时触发
Application_BeginRequest	每次页面请求开始时触发(理想情况下是在页面加载或刷新时)
Application_EndRequest	每次页面请求结束时(即每次在浏览器上执行页面时)触发
Session_Start	每次新的会话开始时触发
Session_End	会话结束时触发

4. Global.asax 文件

每个 Web 网站都对应一个 Global.asax 配置文件。Global.asax 文件包含了所有应用程序的配置设置,并存储了所有的事件处理程序。Global.asax 文件存放在网站的根目录下。Application 对象和 Session 对象的所有事件都存放在 Global.asax 文件中。

默认的 Global.asax 文件内容为:

```
<%@ Application Language = "C#" %>
<script runat = "server">
    void Application_Start(object sender, EventArgs e)
    {
        //在应用程序启动时运行的代码
    }
    void Application_End(object sender, EventArgs e)
    {
        //在应用程序关闭时运行的代码
    }
    void Application_Error(object sender, EventArgs e)
    {
        //在出现未处理的错误时运行的代码
    }
    void Session_Start(object sender, EventArgs e)
    {
        //在新会话启动时运行的代码
    }
    void Session_End(object sender, EventArgs e)
    {
        //在会话结束时运行的代码
        /* 注意: 只有在 Web.config 文件中的 sessionstate 模式设置为
          InProc 时,才会引发 Session_End 事件.如果会话模式 */
        //设置为 StateServer 或 SQLServer,则不会引发该事件
    }
</script>
```

当用户请求启动应用程序并创建新的会话时，首先触发 Application_OnStart 事件，然后才是 Session_OnStart 事件。当处理完当前所有请求之后，服务器首先对每个会话调用 Session_OnEnd 事件，删除所有的活动会话，释放占用的系统资源，然后调用 Application_OnEnd 事件关闭应用程序。

注意：在事件的处理程序代码中，不能包含任何输出语句，因为 Global.asax 文件只能被调用，不会显示在页面上。

5. Global.asax 文件应用实例——网站访问人数统计

在网站的首页上，经常会看到网站统计的当前在线人数。要实现这样的功能很简单，需要在 Global.asax 中为应用程序启动事件添加有关的代码。

例 8-7 使用 Global.asax 文件统计在线访问人数。

[Global.asax]

```
<%@ Application Language = "C#" %>
<script runat = "server">
    //当应用程序启动时,设置全局变量 VistorCount 为 0
    protected void Application_Start(object sender,EventArgs e)
    {
        Application["VistorCount"] = 0;
    }
    protected void Application_End(object sender, EventArgs e)
    {
        // 应用程序关闭时运行的代码
    }
    //当会话开始时,在线人数值加 1; 并设定会话超时时限为 2 分钟
    protected void Session_Start(object sender,EventArgs e)
    {
        Application.Lock( );
        Application["VisitorCount"] = (int)Application["VisitorCount"] + 1;
        Application.UnLock( );
        Session.Timeout = 2;
    }
    //当会话结束时,在线人数值减 1
    protected void Session_End(object sender,EventArgs e)
    {
        Application.Lock( );
        Application["VisitorCount"] = (int)Application["VisitorCount"] - 1;
        Application.UnLock( );
    }
</script>
```

本例中，使用全局可访问的 Application 对象存储在线人数，为了避免在同一时间多个用户访问网站并修改计数器值，采用了 Application 对象的加锁和解锁方法。

当任何一个用户登录网站时，在 Session_OnStart 事件里让在线人数加 1；当用户离开时，在 Session_OnEnd 事件里在线人数减 1。这样用户无论如何刷新网页，在线人数都不会改变。这里要注意的是，当一个用户关闭浏览器时，Session_End 并不会马上发生，而是要等待一个指定的时间（由 Session 对象的 TimeOut 属性指定，默认为 20 分钟）后才触发

Session_End 事件。当一个会话超时后，用户再次访问网站，则会开启一个新的会话。

在页面上，只要有如下代码即可：

```
protected void Page_Load(object sender, EventArgs e)
{
    string info = " 目前在线人数为: {0}";
    info = string.Format(info, Application["VisitorCount"]);
    lblInfo.Text = info;
}
```

程序的运行结果如图 8-5 所示。

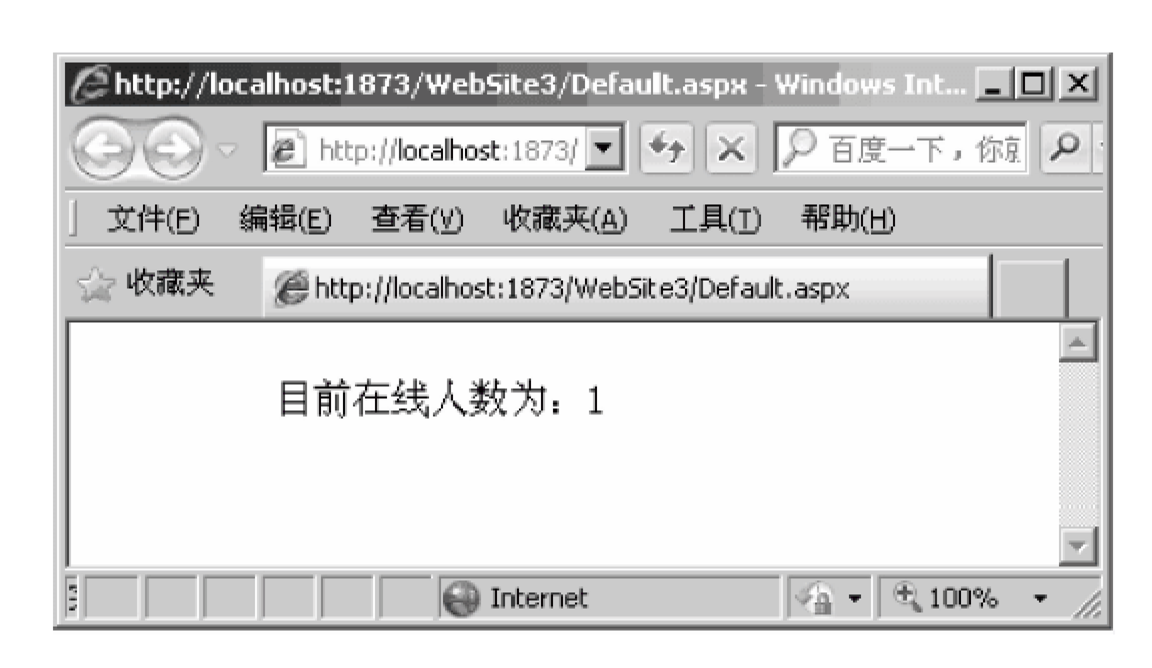


图 8-5 Global.asax 文件应用实例

8.2.2 Session 对象

通常来说，用户访问网站就被视为"用户与服务器进行了一次会话"，Session 就是用于保存会话信息的对象。同一个 Web 服务器可能同时被多个用户访问，每个用户都与服务器建立一个"会话"关系。也就是说，从用户到达某个特定主页开始，一直到关闭浏览器的那段时间，每个用户都会单独获得一个 Session 对象。

Session 对象属于 HttpSessionState 类。Session 对象记载了特定客户的信息，并且这些信息只能由客户自己使用，不能被其他用户访问。换句话说，就是在同一个用户访问的不同页面间可以共享 Session 数据，但是在不同用户间不能共享数据。

为了方便管理，服务器给每个用户都分配一个唯一的标识符，即 SessionID，这样服务器就能够识别来自同一用户的一系列请求了。如图 8-6 所示，当用户第一次请求一个 Web 页面时，服务器创建一个 Session(记录了 Session 变量 name=William)，同时分配给该用户一个 SessionID，并通过 Cookie 发送到客户端。当该用户再次请求另一个页面时，必须同时加载上自己的 SessionID。服务器收到请求后，就搜索与那个 ID 匹配的 Session，找到请求的 Session 变量返回给用户，这样 Session 就从一个页面传递到下一个页面，服务器也就能够识别来自同一用户的连续请求了。

要说明的是，Session 保存在服务器端，Cookies 则保存在客户端，Session 的工作机制要用到 Cookies。如果客户端浏览器不支持 Cookies 或者关闭了 Cookies，Session 也就无法使用了。

下面对 Session 对象的属性、方法和事件等进行介绍。

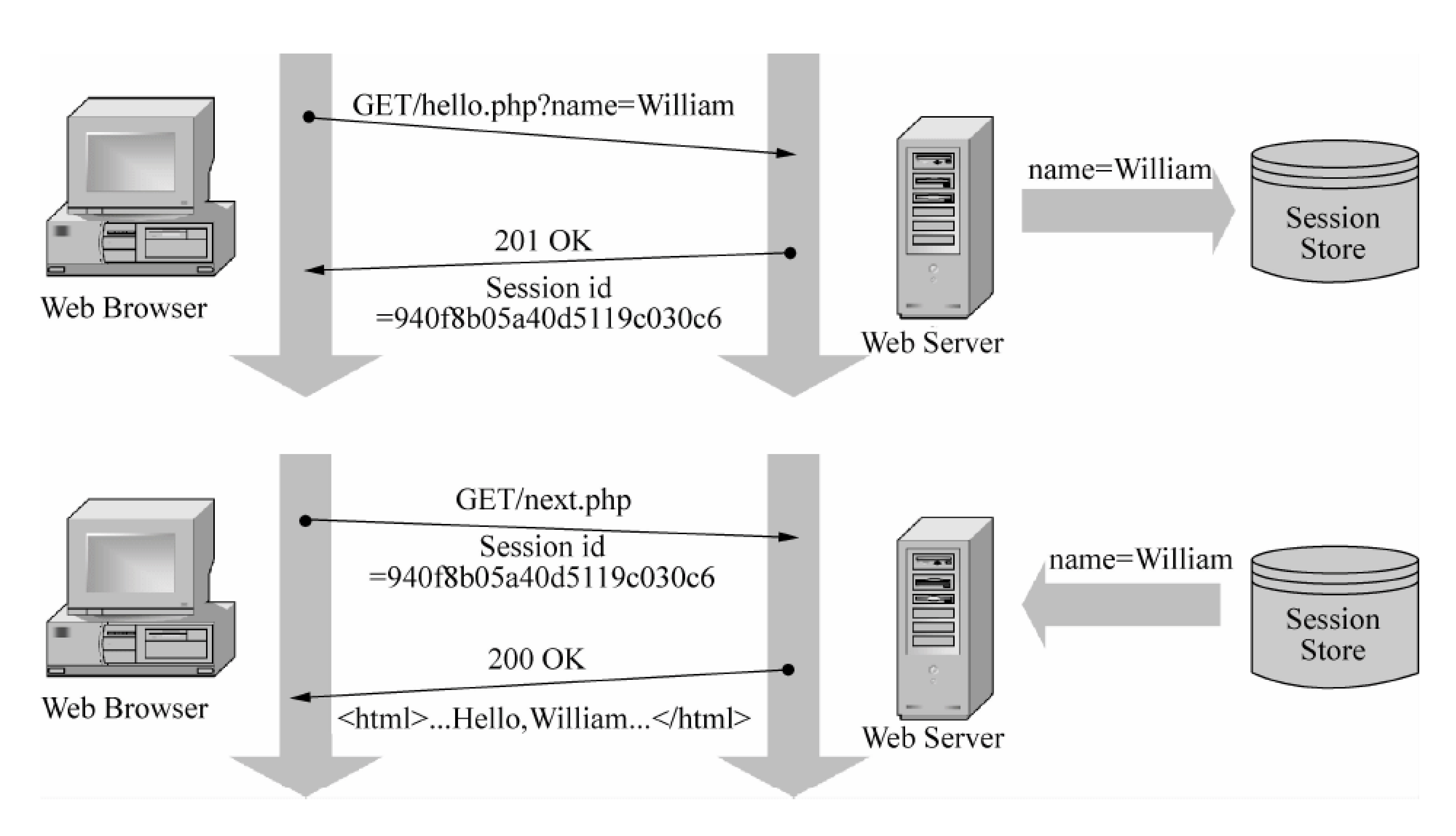


图 8-6　Web 上的 Session 管理

1. Session 对象的属性

Session 对象的主要属性如表 8-11 所示。

表 8-11　Session 对象的属性

属　　性	说　　明
SessionID	获取会话的唯一标识符(只读,长整型)
Timeout	获取和设置会话时间的超时时限,默认值为 20 分钟
Count	获取会话状态集合中的项数
Item	获取或设置会话值的名称
IsNewSession	若该会话是由当前请求创建的,该属性将返回值 true

当用户第一次访问一个网站时,服务器就给该用户建立了一个 Session 对象,并分配一个唯一的 SessionID。Session 对象所创建的变量如同全局变量一样,在该用户访问的每个 Web 页面程序中都可以直接读取。

创建一个 Session 对象和给 Session 变量赋值的语法是一样的。第一次给 Session 变量赋值即自动创建 Session 对象,以后再赋值就是修改其中的值了。语法如下:

```
Session ["键名"] = 值
```

Session 对象有自己的有效期。在有效期内,如果客户端不再向服务器发出新请求或刷新页面,该 Session 就会自动结束并释放出占用资源,即 Session 变量的值被清空。通过 TimeOut 属性可以设置 Session 对象的超时时间。

例如:

```
Session.Timeout = 90        //将有效期设置为 90 分钟
```

2. Session 对象的方法

Session 对象的主要方法如表 8-12 所示。

表 8-12　Session 对象的方法

方　　法	说　　明
Abandon	清除用户的 Session 对象,释放系统资源。调用该方法后会触发 session_OnEnd 事件
Add	添加一个新项到会话状态中
Clear	清除当前会话状态的所有值
CopyTo	将当前会话状态值的集合复制到一个一维数组中
Remove	删除会话状态集合中的项
RemoveAt	删除会话状态集合中指定索引处的项
RemoveAll	清除会话状态集合中的所有的键和值

用户在一个网站内浏览 Web 页面的整个过程期间,称为一个“Session 期间”。在一个 Session 期间内(未超时之前),如果使用了 Abandon 方法,可以清除存储在 Session 中的所有对象和变量,结束该会话并释放系统资源。如果不使用 Abandon 方法,系统将一直等到 Session 超时才将 Session 中的对象和变量清除。

例 8-8　使用 Session 对象统计某一用户的访问次数。

```
protected void Page_Load(object sender, EventArgs e)
{
    //创建 Session 变量 username
    Session["username"] = "jyu" ;
    //创建 Session 变量 visits
    Session["visits"] = Convert.ToInt32(Session["visits"]) + 1;

    String StrName = Session["username"].ToString();
    Response.Write ("你好! " + StrName) ;
    Response.Write ("<br>欢迎你的第 " + Session["visits"] + " 次访问");
}
```

程序运行的结果如图 8-7 所示。



图 8-7　使用 Session 对象统计某一用户的访问次数

3. Session 对象的事件

Session 对象有 Session_OnStart 事件和 Session_OnEnd 事件。

Session_OnStart 事件在用户与服务器创建一个新的会话时触发,服务器在执行请求的页面之前先处理该脚本。可以在该事件中定义所有在页面中需要使用的 Session 变量。

Session_OnEnd 事件在 Session 结束时被调用。当程序调用了 Session 对象的

Abandon 方法或发生超时情况时，触发 Session_OnEnd 事件。Session_OnEnd 事件一般用于清理系统对象或变量的值，释放系统资源。

这两个事件的代码都存储在服务器根目录下的 Global. asax 文件中。

8.2.3 Cookie 对象

Session 对象能够保存用户信息，但是 Session 对象并不能够持久的保存用户信息，当用户在限定时间内没有任何操作时，用户的 Session 对象将被注销和清除，在持久化保存用户信息时，Session 对象并不适用。

使用 Cookie 对象能够持久化的保存用户信息，相比于 Session 对象和 Application 对象而言，Cookie 对象保存在客户端，而 Session 对象和 Application 对象保存在服务器端，所以 Cookie 对象能够长期保存。Web 应用程序可以通过获取客户端的 Cookie 识别和判断一个用户的身份。

ASP.NET 提供了 HttpCookie 对象处理 Cookie，该对象是 System. Web 命名空间中的 HttpCookie 类的对象。每个 Cookie 都是 HttpCookie 类的一个实例。

1. Cookie 对象的属性

Cookie 是一个很小的文本文件，由网站服务器在用户第一次访问时生成，发送到客户端的硬盘里，用来存储用户的特定信息。当用户再次访问该站点时，浏览器就会在本地硬盘上查找与该网站相关联的 Cookie。如果存在，就将它与页面请求一起发送到网站服务器，服务器上的 Web 应用程序就可以读取 Cookie 中包含的信息。

一般来说，每个客户端最多能存储 300 个 Cookie，一个站点能为一个单独的用户最多设置 20 个 Cookie。

Cookie 对象的主要属性如表 8-13 所示。

表 8-13　Cookie 对象的主要属性

属　　性	说　　明
Domain	获取或设置 Cookie 应属于的域名
Expires	获取或设置 Cookie 的过期时间
Name	获取或设置 Cookie 的名称
Path	获取或设置当前 Cookie 适用的路径
Secure	指定是否通过 SSL(即仅通过 HTTPS)传输 Cookie
Value	获取或设置 Cookie 的 Value
Values	获取在 Cookie 对象中包含的键值对的集合

2. Cookie 对象的方法

Cookie 对象的方法如表 8-14 所示。

表 8-14　Cookie 对象的主要方法

方　　法	说　　明
Add	增加 Cookie
Clear	清除 Cookie 集合内的变量
Get	通过键名或索引得到 Cookie 的值
Remove	通过 Cookie 的键名或索引删除 Cookie 对象

3. 访问 Cookie

ASP.NET 包含两个内部 Cookie 集合：Request 对象的 Cookies 集合和 Response 对象的 Cookies 集合。其中，Request 对象的 Cookies 集合包含由客户端传输到服务器的 Cookie，这些 Cookie 以 Cookie 标头的形式传输。Response 对象的 Cookies 集合包含一些新的 Cookie，这些 Cookie 在服务器上创建并以 Set-Cookie 标头的形式传输到客户端。

浏览器负责管理用户本地硬盘上的 Cookie。在 ASP.NET 页面中，可以通过 Response 对象来创建和设置 Cookie，即向浏览器写入 Cookie。通过 Request 对象可以读取 Cookie。

关于 Cookie 的设置和读取方法，参见 8.1.1 节和 8.1.2 节的有关内容。

例 8-9 设置和读取 Cookie 值。

```
protected void Page_Load(object sender, EventArgs e)
{
    DateTime now = DateTime.Now;
    //创建了一个 Cookie 对象,名为 LastVistTime
    HttpCookie MyCookie = new HttpCookie("LastVistTime");

    //设置 Cookie 值为当前时间
    MyCookie.Value = now.ToString();
    //设置 Cookie 超时时间为 2 个小时
    MyCookie.Expires = now.AddHours(2);
    //将新 Cookie 添加到 Response 对象的 Cookies 集合中
    Response.Cookies.Add(MyCookie);

    //从 Request 对象的 Cookie 集合中读取名为 LastVistTime 的 Cookie 值
    string myVistTime = Request.Cookies["LastVistTime"].Value;

    Response.Write("上次访问时间为: " + myVistTime);
}
```

程序中，创建 Cookie 的代码可以简写为：

```
HttpCookie MyCookie = new HttpCookie("LastVistTime", now.ToString());
```

或

```
Reponse.Cookies["LastVistTime"] = now.ToString();
```

程序的运行结果如图 8-8 所示。



图 8-8 设置和读取 Cookie 值

注意：由于 Cookie 存储在客户端，所以不能在服务器端编程直接修改 Cookie。如果确实需要修改的话，就创建一个同名的 Cookie，然后发送到客户端覆盖原 Cookie。

当删除 Cookie 时，则可以利用 Cookie 的 Expires 属性，创建一个与原 Cookie 同名的 Cookie，设置 Expires 属性为过去的某一天，将其发送到客户

端覆盖原Cookie。当浏览器检查Cookie的失效日期时,就会删除这个过期的Cookie。

8.2.4 ViewState对象

在针对同一页面的多次请求时,可以使用ViewState对象保存服务器控件的状态信息。简单地说,ViewState就是用于维护页面的UI状态的。

与Session对象相比较,Session对象保存在服务器内存,大量使用Session会使得服务器负担加重。ViewState对象将数据存入到页面隐藏控件中,不占用服务器资源。Session的默认超时时间是20分钟,而ViewState则永远不会超时。ViewState只能在同一个页面的多次回发间保存状态信息,不能解决在多个页面间共享状态信息的问题,而后者可通过Session对象解决。

跟隐藏控件相似,ViewState在同一个页面的多次请求间进行值传递。这是因为一个事件发生后,页面可能会刷新,如果定义全局变量会被清零,而使用ViewState对象则可以保存数据。因此,所有Web服务器控件都使用ViewState在页面回发期间保存自己的状态信息。如果某个控件不需要在回发期间保存状态,最好关闭它的ViewState,避免不必要的资源浪费。通过给@Page指令添加EnableViewState=false属性可以禁止整个页面的ViewState。

ViewState对象保存数据采用"键Key—值Value"对的形式。格式如下:

```
ViewState["键名"] = 数据值;
```

可以用ViewState["键名"]取出保存在ViewState对象中的数据。注意,这时取出的数据类型为Object,必要时要转换成特定的数据类型。

例如,使用ViewState对象保存信息的代码如下:

```
//保存在ViewState对象中
ViewState["SortOrder"] = "DESC";
//从ViewState中读取
String sortOrder = (string)ViewState["SortOrder"];
```

例 8-10 用ViewState记录同一个页面中按钮被单击的次数。

```
protected void btnClick_Click(object sender, EventArgs e)
{
    int counter;                              //计数器
    if (ViewState["Counter"] == null)
    {
        //如果是第一次单击按钮
        counter = 1;
    }
    else
    {
        //从ViewState对象中取出上次保存的counter变量值,并累加1
        counter = (int)ViewState["Counter"] + 1;
```

```
        }
        ViewState["Counter"] = counter;          //将 counter 变量值保存到 ViewState 对象中
        lblInfo.Text = "您单击了" + counter.ToString() + "次按钮.";
    }
```

上述代码中，使用变量 counter 作为按钮单击次数的计数器，并将 counter 变量值保存到 ViewState["Counter"]中。在同一个页面中多次单击按钮，运行结果如图 8-9 所示。



图 8-9 用 ViewState 记录同一个页面中按钮被单击的次数

8.3 习题和上机练习

1. 填空题

(1) Response 对象的________方法可以使得浏览器显示另外一个 URL。

(2) Server 对象的________方法可以将虚拟路径转化为物理路径。

(3) 设置 Cookie 采用________对象，读取 Cookie 采用________对象。

(4) Server 对象的 ScriptTimeout 属性的默认值是________，Session 对象的 Timeout 属性的默认值是________。

(5) Request 对象的________集合可以用来获取服务器的名称。

(6) ________对象在同一个页面的多次回发间保存状态信息，要想在同一网站的多个页面间共享信息，需使用________对象。

2. 选择题

(1) Request.Form("username")中的 username 是(　　)。

A. 表单的名称　　B. 网页的名称

C. 表单元素的名称　　D. 表单按钮的名称

(2) 不需要在网页第一行添加<% Response.Buffer=True %>的是(　　)。

A. Response.Redirect　　B. Response.Clear

C. Response.End　　D. Response.Flush

(3) 有如下 URL：http://127.0.0.1/test.aspx? user=aa，如果想接收 user 中的内容，以下正确的是(　　)。

A. Request.Form("user")　　B. Request.Querystring("user")

C. Request.Cookies("user")　　D. Request.ServerVariables("user")

(4) 如果要获得服务器的 IP 地址，应使用下面(　　)语句。

A. Request. ServerVariables(“LOCAL_ADDR”)

B. Request. ServerVariables(“REMOTE_ADDR”)

C. Request. ServerVariables(“REMOTE_HOST”)

D. Request. ServerVariables(“URL”)

(5) 如果想在 URL 里带有汉字参数,下面正确的是(　　)。

A. <a href=test. asp? hz=<%=Server. HTMLencode("你好")%>>问候

B. <a href=test. asp? hz=<%=Server. URLencode("你好")%>>问候

C. <a href=test. asp? hz=<%=Server. MapPath("你好")%>>问候

(6) 要在网页中输出网易,正确的是(　　)。

A. Response. Write(“网易”)

B. Response. Write(Server. URLencode(“网易”))

C. Response. Write(Server. HTMLencode(“网易”))

D. 以上都不对

3. 问答题

(1) Application 对象和 Session 对象的区别是什么?

(2) Session 对象和 ViewState 对象的区别是什么?

(3) 简述 Session 和 Cookie 的区别。

(4) 请分别用 HTML、JavaScript、C#、ASP.NET 语句输出“祝你好运”这句话。

4. 上机练习

(1) 设计一个用户登录页面,要求输入账号和密码,并单击“登录”按钮。如果输入的账号为 abc,密码是 123word,则跳转到另一个网页并显示“欢迎访问”;否则,在当前页面输出“账号或密码不正确”。

(2) 将上题稍加修改,在输入正确的账号和密码,输出“你是第 n 次访问本站”,中间的 n 在刷新网页时也要同时刷新。

第 9 章　访问 Web 数据库

目前，大量应用程序都是数据驱动的，都离不开数据库技术的支持，需要频繁的读取、显示以及更新数据。.NET 框架中提供了多种方式来访问数据存储，ADO.NET 是最直接的方式，也是最灵活、执行效率最高的方式。本章主要讲解 ADO.NET 的体系结构以及核心对象的使用。

9.1　ADO.NET 体系结构

ADO.NET 是一组向 .NET 程序员公开数据访问服务的类，为创建分布式数据共享应用程序提供了一组丰富的组件，提供了对关系数据库、XML 数据、OFFICE 文档数据等多种数据存储的访问方式。

ADO.NET 采用多层体系结构，其核心组件结构如图 9-1 所示。

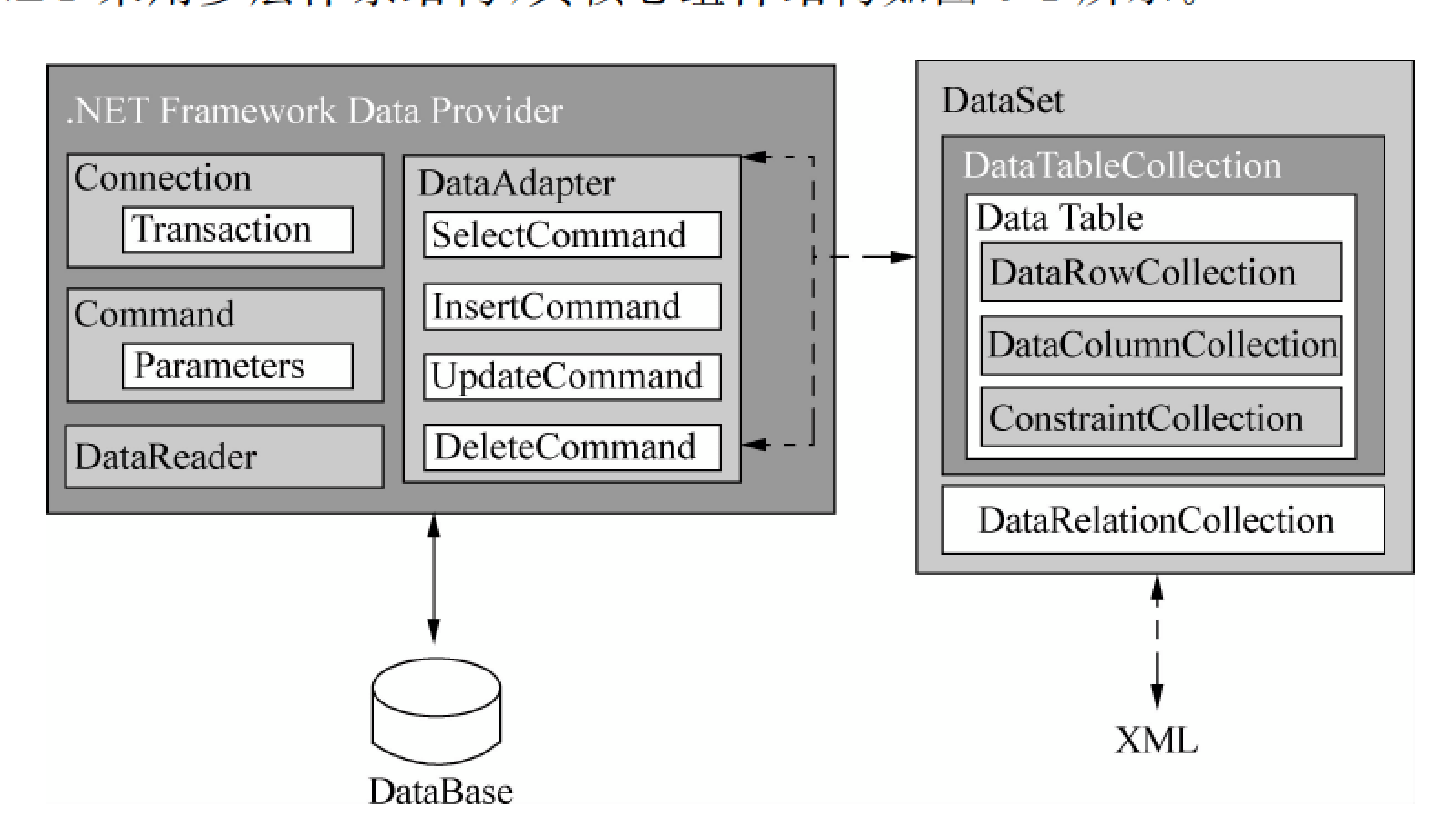


图 9-1　ADO.NET 核心组件结构

可以看出，ADO.NET 用于访问和操作数据的两个主要组件是 .NET Framework 数据提供程序和 DataSet。前者用于连接到不同的数据存储，实现在线的数据检索和更新操作；后者则代表数据存储的内存映像，可以将关系数据及 XML 数据加载到内存中，然后断开与数据源的连接，离线进行各种处理，最后再将更新一次性的保存到数据源中。

9.1.1　ADO.NET 数据提供程序

数据提供程序是应用程序与数据源之间的一座桥梁，包含一组用于访问特定数据库，执

行 SQL 语句并获取值的.NET 类，主要有：

- Connection 类：使用它建立与数据源的连接。
- Command 类：使用它执行 SQL 命令及存储过程。
- DataReader 类：提供对 Select 语句查询结果的快速、只读、只进的访问方法。
- DataAdapter 类：作为数据源与 DataSet 之间的桥梁，可以从数据源获取数据填充到 DataSet 中，也可以依照 DataSet 中的修改更新数据源。

早期的 ADO 技术中使用通用的数据提供程序处理各种不同的数据源，而 ADO.NET 改变了这种做法，为不同的数据源设计了不同的数据提供程序。.NET Framework 共设计了 4 种数据提供程序。

- SQL Server 提供程序：提供对 SQL Server 数据库的优化访问。
- Oracle 提供程序：提供对 Oracle 数据库的优化访问。
- OLE DB 提供程序：提供对 OLE DB 驱动的任意数据库的访问，以往的多数数据库产品都支持 OLE DB 数据访问。
- ODBC 提供程序：提供对 ODBC 驱动的任意数据库的访问。

图 9-2 显示了 ADO.NET 数据提供程序的模型结构。在为应用程序选择数据提供程序时，应尽量选择为数据源定制的.NET 提供程序，若找不到合适的定制提供程序，再选择基于 OLE DB 的提供程序，在极少数情况下，若依然找不到合适的 OLE DB 提供程序，最后可以选择 ODBC 提供程序。

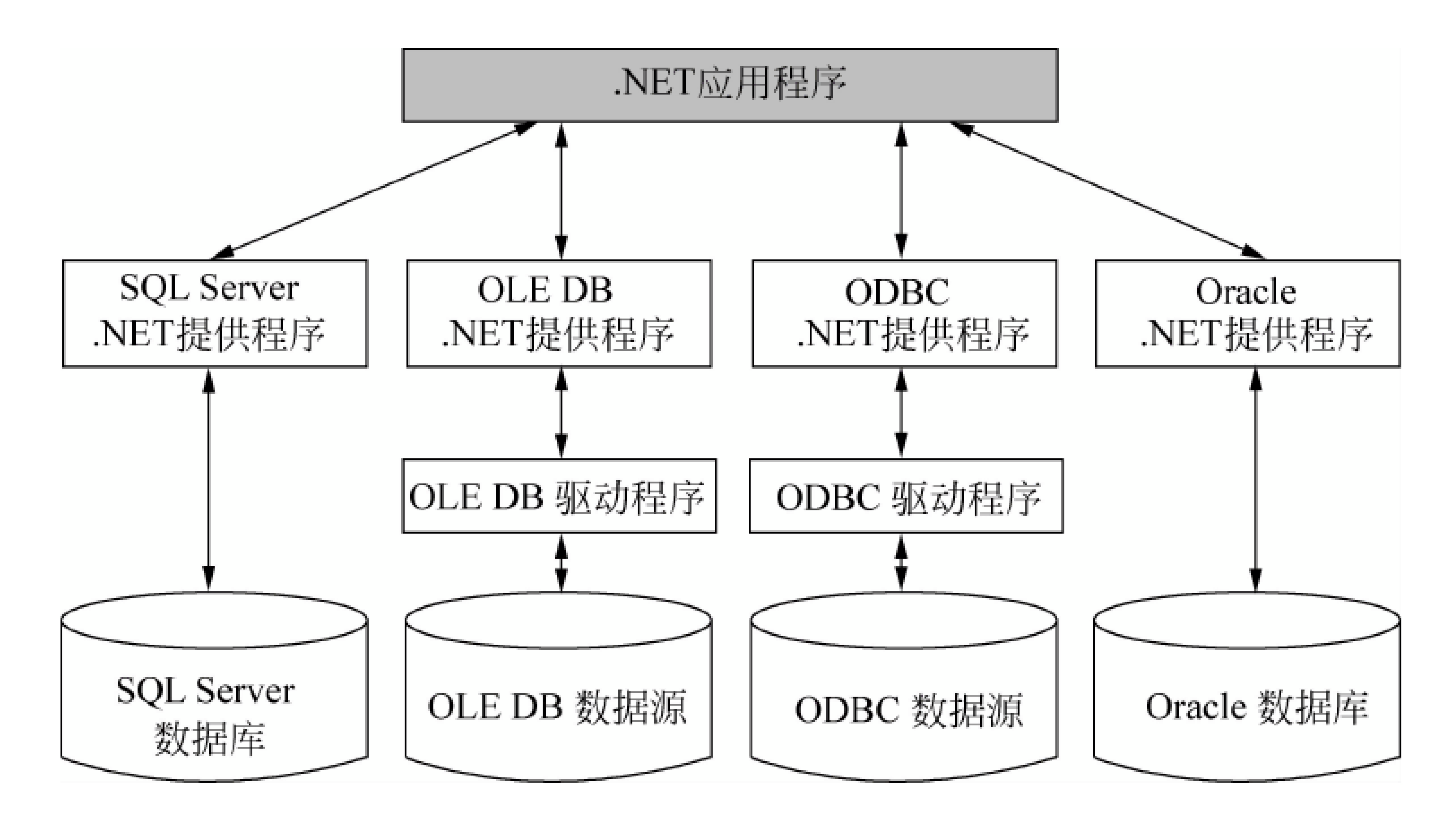


图 9-2　ADO.NET 数据提供程序模型结构

9.1.2　ADO.NET DataSet

DataSet 是数据驻留在内存中的表示形式，不管数据源是什么，都可提供一致的关系编程模型。DataSet 表示包括相关表、表间关系及约束在内的整个数据集，其对象结构如图 9-3 所示。

DataSet 中的核心对象简单介绍如下：

- DataTableCollection：数据表的集合。DataSet 中可以包含 0 个到多个数据表，DataTableCollection 包含了 DataSet 中所有的 DataTable 对象。

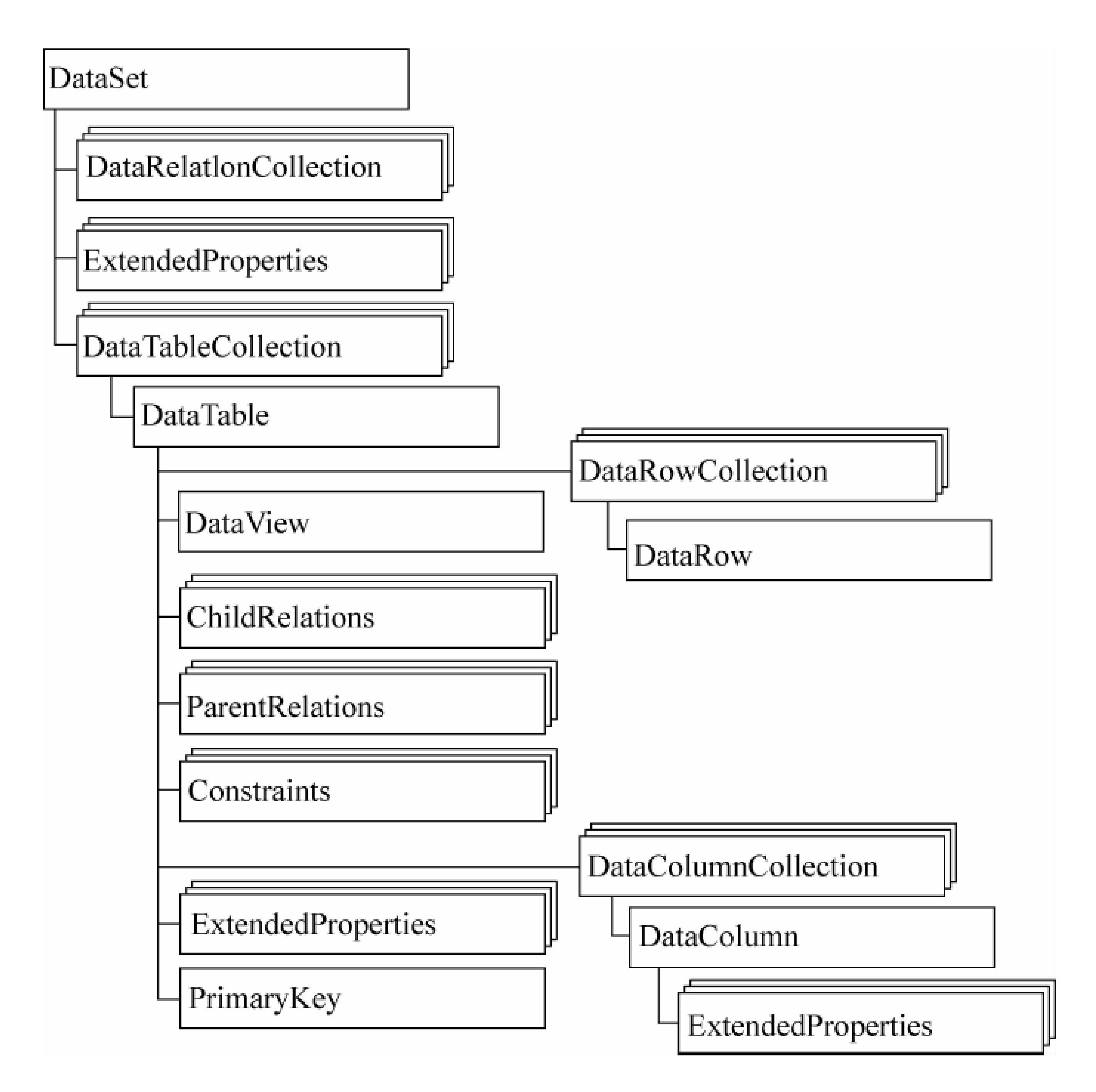


图 9-3 DataSet 对象结构

- DataRelationCollection：关联的集合。在关系数据库中，表和表之间存在一定的关联（如外键），DataSet 作为数据库的内存映像，也可以在内存表之间建立关联。DataRelationCollection 包含了 DataSet 中所有的 DataRelation 对象。
- DataTable：数据表，或称内存表，表示内存驻留数据的单个表。DataTable 中包含由 DataColumnCollection 表示的列的集合以及由 Constraints 表示的约束集合，这两个集合共同定义了数据表的架构。DataTable 中还包含由 DataRowCollection 所表示的行的集合，即表的数据。需要说明的是，既可以在 DataSet 中创建内存表，也可以脱离 DataSet 创建单独的内存表使用。
- DataView：数据视图，代表存储在 DataTable 中数据的不同表现形式。通过 DataView，可以为表中的数据进行排序，或定制筛选器以过滤数据。
- DataRow：数据行，代表着关系表中的一行数据。可以对行中的各个数据项按顺序号或列名称进行访问。
- DataColumn：数据列，代表着关系中的一个属性，通过定义 DataColumn 定义关系表的结构。
- PrimaryKey：主键，对于内存表，也可以将它的一个或多个数据列定义为主键，以实现完整性约束，或者在内存表中索引查找指定的数据行。

9.1.3 ADO.NET 类的组织

ADO.NET 类分组保存在多个命名空间中，每个提供程序都有自己的命名空间，通用类保存在 System.Data 命名空间。表 9-1 介绍了这些命名空间。



表 9-1 ADO.NET 常用类的组织

命名空间	描　　述
System.Data	包含关键的容器类，这些类为关系、表、视图、数据集、行和约束建立模型，另外还包含了基于连接的数据对象要实现的关键接口的定义
System.Data.Common	包含由各种 .NET Framework 数据提供程序共享的抽象类，具体的数据提供程序继承这些类，创建自己的版本
System.Data.OleDb	包含用于连接 OLE DB 数据提供程序的类，主要有 OleDbCommand、OleDbConnection、OleDbDataReader 及 OleDbDataAdapter 等
System.Data.SqlClient	包含用于连接微软 SQL Server 数据库所需的类，包括 SqlDbCommand、SqlDbConnection、SqlDbDataReader 及 SqlDbDataAdapter 等，这些类使用经过优化的 SQL Server 的 TDS 接口
System.Data.OracleClient	包含连接 Oracle 所需的类，包括 OracleCommand、OracleConnection、OracleDataReader 及 OracleDataAdapter 等，这些类使用经过优化的 Oracle 的 OCI 接口
System.Data.Odbc	包含连接大部分 ODBC 驱动所需的类，包括 OdbcCommand、OdbcConnection、OdbcDataReader 及 OdbcDataAdapter 等

9.2 使用 ADO.NET 数据提供程序访问数据库

9.2.1 访问数据库的一般方法

使用.NET 数据提供程序访问数据库的一般步骤如下：

(1) 使用 Connection 对象建立到数据源的连接。

(2) 在连接之上建立 Command 对象，通过它向数据库发送 SQL 命令。

(3) 接收 SQL 命令的返回结果。若返回记录集，有两种处理方法：一是采用 DataReader 对象在线逐条获取数据，二是采用 DataTable 对象一次性获取所有数据到内存进行离线处理。

(4) 释放数据库操作对象，并关闭数据库连接。

ADO.NET 针对不同的数据源设计有不同的数据提供程序，应根据实际应用环境选择合适的类来使用。下面以查询 SQL Server 数据库为例说明基本使用方法。

例 9-1 连接 Pubs 数据库，查询 Authors 表中的所有记录并在页面上列表显示。

启动 VS2008，建立 Web 应用程序项目，取名 dbprj。使用可视化工具建立 Web Form 页面 exam9_1.aspx，并在工具箱的数据选项卡上选择 GridView 控件添加到页面上，最后生成的页面代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="exam9_1.aspx.cs" Inherits
="dbprj.exam9_1" %>
<html>
<head runat="server">
    <title>ADO.NET 示例</title>
</head>
<body>
    <form id="form1" runat="server">
```

```
            <asp:GridView ID = "GridView1" runat = "server"></asp:GridView>
        </form>
    </body>
    </html>
```

在 Web 页的设计视图,双击页面空白区域,进入代码窗口,编写如下代码:

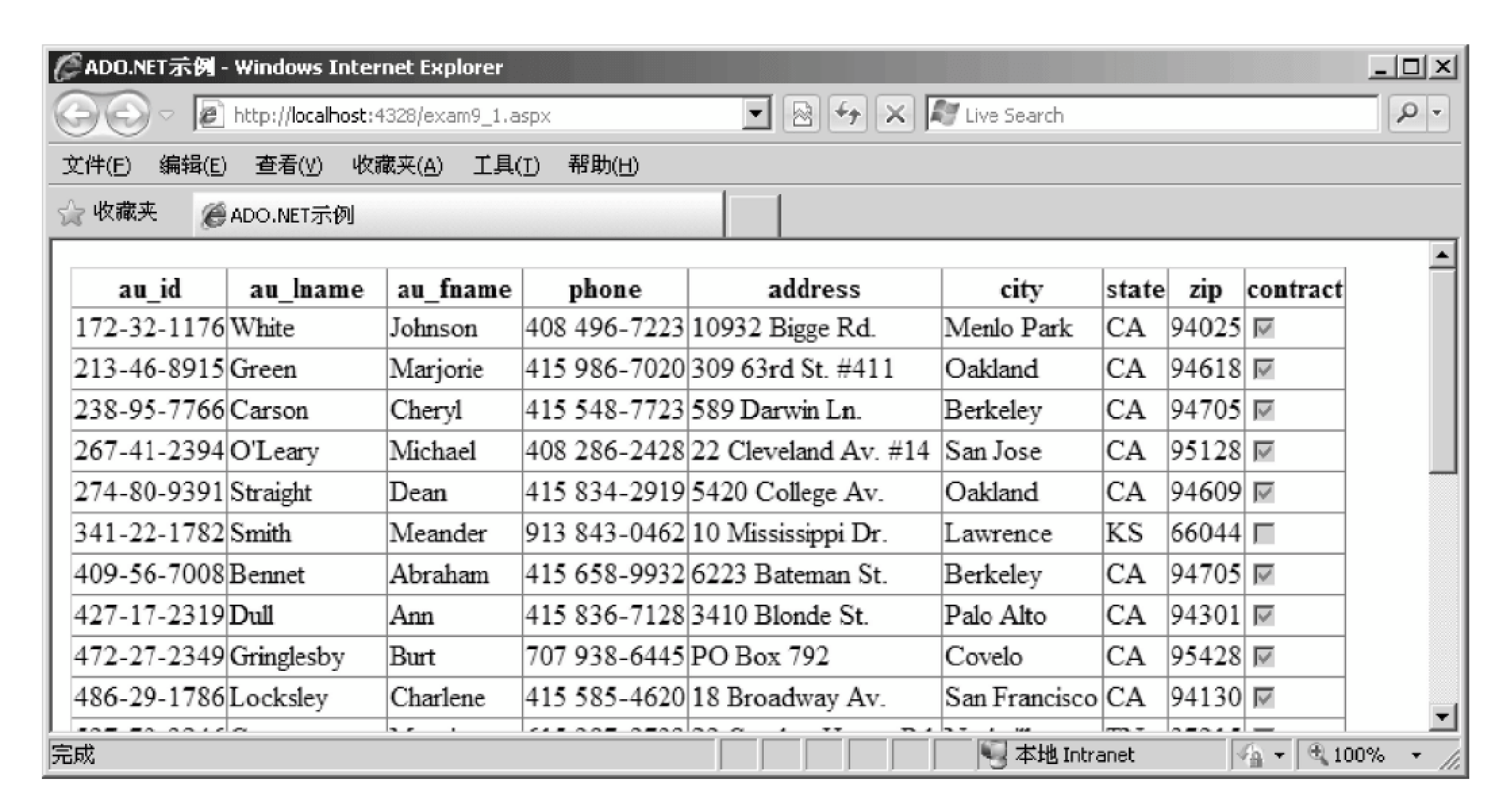
```
using System;
using System.Data.SqlClient;
namespace dbprj
{
    public partial class exam9_1 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            string connstr = @"Data Source = .\sqlexpress;Initial Catalog = pubs;Integrated
Security = True";
            SqlConnection conn = new SqlConnection(connstr);
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = conn;
            cmd.CommandText = "select * from authors";
            try
            {
                conn.Open();
                SqlDataReader dr = cmd.ExecuteReader();
                GridView1.DataSource = dr;
                GridView1.DataBind();
                dr.Close();
            }
            catch{ }
            finally
            {
                conn.Close();
            }
        }
    }
}
```

保存项目,在解决方案资源管理器窗口中右击 exam9_1.aspx 页面,选择"在浏览器中查看"项访问该页面,运行结果如图 9-4 所示。

本例中的核心问题说明如下:

(1) 由于要操作 SQL Server 数据库,可以使用 SQL Server .NET 提供程序,也可以使用 OLE DB .NET 提供程序,甚至还可以使用 ODBC .NET 提供程序。从执行效率来看,首选 SQL Server .NET 提供程序,涉及 SqlConnection、SqlCommand 及 SqlDataReader 等类,这些类都位于 SqlClient 命名空间下,所以在代码开头先导入了该命名空间。

(2) 本例中使用 SqlConnection 对象连接数据库,在创建连接对象时,需要指明数据库的位置以及登录验证信息等,这些信息通过连接字符串指定。关于连接字符串的格式将在



au_id	au_lname	au_fname	phone	address	city	state	zip	contract
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025	☑
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	☑
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705	☑
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128	☑
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA	94609	☑
341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS	66044	☐
409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA	94705	☑
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301	☑
472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA	95428	☑
486-29-1786	Locksley	Charlene	415 585-4620	18 Broadway Av.	San Francisco	CA	94130	☑

图 9-4 查询并显示 Authors 表中的所有记录

后续章节中详细介绍。调用连接对象的 Open 方法可以打开连接，使用完成后，再调用其 Close()方法关闭连接。

(3) 使用 SqlCommand 对象向数据库发送了一条 Select 语句，执行后将返回一个记录集，本例使用 SqlDataReader 对象操作该记录集，这种方式占用内存资源很少，执行效率极高。

(4) 本例使用了富数据控件 GridView 显示查询结果。GridView 是.NET 中最重要的一种数据绑定控件，功能强大，使用方便，详细用法将在后续章节中介绍。

(5) 在访问数据库时可能会出现各种各样的异常，所以一定要注意捕获和处理异常。

9.2.2 使用 Connection 对象

ADO.NET 使用 Connection 对象建立到数据源的连接，针对不同的数据提供程序要建立不同的连接对象。例如，使用 SQL Server .NET 提供程序连接 SQL Server 数据库时，应使用 SqlConnection 对象；使用 OLE DB .NET 提供程序连接 SQL Server 数据库时，应使用 OleDbConnection 对象。连接建立后，就可以通过它与数据源交互，执行各项操作；当所有操作完成后，切记要及时释放连接，为后续的请求处理留出宝贵资源；为提高建立连接的效率，通常会使用连接池来缓存和共享连接。

1. 连接字符串

创建连接对象时需要提供连接字符串，它是用分号分隔的一系列名称/值对的选项集合，提供了连接所需要的基本信息，各个项的排列位置及字母大小写没有影响。针对不同的关系数据库及不同的数据提供程序，连接字符串的格式会有所区别，但任何一种连接一般都包含如下几个选项：

1) 服务器位置

指定数据库服务器的名称或地址。该项的名称通常使用 DataSource、Server、Address、

NetWork Address 等，该项的值通常填服务器的 IP 地址或主机名，也可以填 localhost 或 local 代表本地数据库。

2）数据库名称

指定要连接到的数据库实例名。该项的名称通常是 Initial Catalog，或者 DataBase。

3）身份验证方式

可选择操作系统集成的身份验证方式或基于用户名和口令的验证方式。当 ASP.NET 应用程序和数据库服务器位于同一台计算机上时，尽量选择集成验证方式以便建立可信连接(Trusted Connection)；否则，就需要在数据库服务器中为应用程序建立一个数据库用户，提供用户名和口令供验证。建立可信连接通常使用 Integrated Security＝true 或 Trusted Connection＝true 参数；若建立非可信连接，则需要使用 User ID 或 UID 项指定用户名，并通过 Password 或 pwd 项指定验证口令。

例 9.1 中使用 SqlClient 连接到本机上的 SQL Server 服务器，数据库实例名为 pubs，采用 Windows 集成的身份验证方式，其连接字符串的格式为：

```
Server = localhost; Initial Catalog = Pubs; Integrated Security = true
```

若使用 SQL Server 验证，用户名和口令都是 examdbuser，则连接字符串格式为：

```
Server = localhost; Initial Catalog = Pubs; User ID = examdbuser; Password = examdbuser
```

2. 建立连接

连接到不同的数据库需要使用不同的连接对象及连接字符串，下面通过一些实例演示常用的数据库连接方式。

1）使用 SqlClient 连接 SQL Server 数据库

```
using System.Data.SqlClient;                          // 导入 SqlClient 命名空间
…
string constr = @"Data Source = (local); Initial Catalog = Northwind; Integrated Security =
true"
SqlConnection conn = new SqlConnection( constr );     // 建立连接对象
```

2）使用 OracleClient 连接 Oracle 数据库

```
using System.Data.OracleClient;                         // 导入 OracleClient 命名空间
…
string constr = @" Data Source = orcl;User ID = scott;Password = tiger"
OracleConnection conn = new OracleConnection(connstr);  // 建立连接对象
```

3）使用 OLE DB 连接 ACCESS 数据库

```
using System.Data.OleDb;                              // 导入 OleDb 命名空间
…
string constr = @"Provider = Microsoft.Jet.OLEDB.4.0; Data Source = D:\dbprj\pubs.mdb"
OleDbConnection conn = new OleDbConnection(connstr);  // 建立连接对象
```

4）使用 OLE DB 连接 SQL Server 数据库

```
using System.Data.OleDb;                                  // 导入 OleDb 命名空间
…
string constr = "Provider=SQLOLEDB; Data Source=.\sqlexpress; Integrated Security=SSPI;
            Initial Catalog=pubs"
OleDbConnection conn = new OleDbConnection(connstr);      // 建立连接对象
```

5）使用 OLE DB 连接 Oracle 数据库

```
using System.Data.OleDb;                                  // 导入 OleDb 命名空间
…
string constr = @"Provider=MSDAORA;Data Source=orcl;User ID=scott;Password=tiger"
OleDbConnection conn = new OleDbConnection(connstr);    // 建立连接对象
```

使用 ADO.NET 还可以连接到 MySQL、FoxPro 等各种数据库服务器或数据库文件，甚至可以将 Excel 工作表作为数据源进行操作，详细的连接方式这里不再过多介绍。

可以看出，连接字符串是建立数据库连接的关键，为正确书写连接字符串，可以借助 VS.NET 提供的服务器资源管理器来自动生成。打开"服务器资源管理器"窗口，右击"数据库连接"，从弹出菜单中选择"添加连接"项，系统将打开如图 9-5 所示的对话框；选择数据源类型，输入服务名，并选择登录到服务器的验证方式(若基于口令验证，则正确输入用户名和口令)，单击对话框底部的"测试连接"按钮，若参数配置争取，系统会提示"测试连接成功"，这时单击"确定"按钮，在服务器资源管理器中将看到新建立的连接；右击该连接，从弹出菜单中选择"属性"项，该连接的详细信息将显示在属性窗口中，如图 9-6 所示，从这里复制连接字符串即可。



图 9-5 "添加连接"对话框

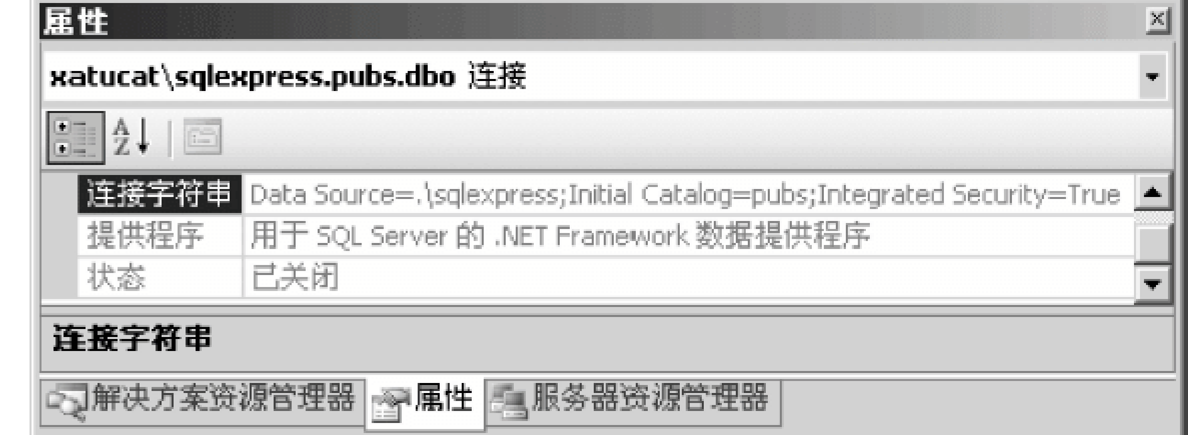


图 9-6 数据库连接的属性

3. 打开和关闭连接

连接对象创建后，需要打开才能真正连接到数据库。调用连接对象的 Open 方法即可打开连接。方法如下：

```
conn.Open( );
```

数据库连接是一种宝贵的服务器资源，用户通常需要花费高额的费用来购买并发连接的授权。在 Web 环境中，当一个线程使用完连接后，应尽快的释放连接，以备其他线程使用；若连接打开后没有及时关闭，将在很长时间内保持占用状态，其他线程将无法使用。

调用连接对象的 Close 方法可以释放连接，代码如下：

```
conn.Close( );
```

调用连接对象的 Dispose 方法也可以释放连接，用法如下：

```
conn.Dispose( );
```

考虑到操作数据库时可能出现各种异常，代码中应尽量使用 try-catch 块捕获并处理异常；另外，连接使用完成后应尽可能释放，好的习惯是在 finally 块中释放连接。这样，一般使用如下的代码框架操作数据库：

```
SqlConnection conn = new SqlConnection(connstr);  // 创建连接对象
try
{
    conn.Open();                                  // 打开到数据库的连接
        …                                         // 操作数据库
}
catch(Exception ex)
{
        …                                         // 处理异常
}
finally
{
        conn.Dispose();                           // 释放连接
}
```

从.NET 2.0 开始引入了 using 语句，可以替代 try-finally 的功能。

例如：

```
using (SqlConnection conn = new SqlConnection(connstr))
{
    conn.Open();
}
```

using 语句定义一个范围，并指定要创建和关闭的对象(此处为 conn 对象)；当代码执行到范围末尾(右大括号的位置)时，或因抛出异常而跳出范围时，系统都将自动清理 using 语句中指定的对象(即 conn 对象)。请看如下的完整示例代码。

例 9-2 连接 Pubs 数据库，查询并显示所有书籍的书号、书名和价格。

```
using System;
using System.Data.SqlClient;
class Program
{
    static void Main()
    {
        string connstr = @ "Data Source = .\sqlexpress; Initial Catalog = pubs;
                           Integrated Security = True";
        string qrystr = " SELECT title_id, title, price from titles "
        using ( SqlConnection conn = new SqlConnection(connstr) )
        {
            SqlCommand cmd = new SqlCommand( qrystr,  conn );
            try
            {
                conn.Open( );
                SqlDataReader reader = cmd.ExecuteReader();
                while (reader. Read())
                {
                    Console.WriteLine ("\t{0}\t{1}\t{2}", reader[0], reader[1], reader[2]);
                }
                reader.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

4. 数据库连接池

建立到数据库的连接是一项复杂而又费时的工作，若每一次操作数据库时都建立连接，使用完成后又释放，必将浪费大量的时间和资源。为提高操作效率，ASP.NET 提供了连接池的机制。在 Web 应用第一次连接数据库时，系统会隐含的建立一定数量的连接，放在池中集中管理；以后每当再次请求该数据库时，直接从池中“借”出一个连接使用，使用完成后再归还到池中，这样可以最大程度地降低重复打开和关闭连接所造成的成本。

Web 服务器通常根据连接字符串自动创建连接池，多个连接若连接字符串相同则使用同一个连接池，若某个连接的连接字符串与现有池的连接字符串不同，系统将创建一个新的连接池；ADO.NET 可同时保留多个池，每种配置各一个。

Web 服务器中的连接池对开发者完全透明，数据访问代码不需要做任何更改。当调用 Open 方法打开连接时，连接实际上由连接池提供而不是再次创建；当调用 Close 方法或 Dispose 方法释放连接时，它并没有真正被释放，而是重新回到了池中等待下一次请求。

5. 保存连接字符串

将连接字符串硬编码到程序中不是好的编程风格，若以后数据库连接参数发生了变化，

所有操作数据库的代码块都需要更改，这将引起灾难性的后果。另外，由于 Web 服务器通常根据连接字符串创建连接池，若不同页面中的连接字符串格式稍有差异，系统将创建不同的连接池，对系统性能也会造成一定影响。

好的编程风格是将连接字符串保存在一个配置文件中，这样每当应用程序要连接数据库时，从配置文件中根据连接字符串的"键名"获取其"键值"，并以此建立连接。若数据库连接参数发生了变化，只需要在配置文件中更改一次"键值"，应用程序中不需要做任何更改。

在 Web 应用中通常将连接字符串保存到主目录下的 Web.config 配置文件中。打开该文件，会看到其中默认有两个配置节，即<appSettings>节和<connectionStrings>节，在这两个配置节下都可以写入连接字符串，一般放在<connectionStrings>中。

例如：

```
<connectionStrings>
  <add name = "connstr" connectionString = "Data Source = .\sqlexpress; Initial Catalog = pubs;
Integrated Security = True"/>
</connectionStrings>
```

或者写在<appSettings>中：

```
<appSettings>
  <add key = "connstr" value = "Data Source = .\sqlexpress; Initial Catalog = Pubs;
     Integrated Security = True"/>
</appSettings>
```

在以上两种方式中，name 和 key 指定连接字符串的"名字"，connectionString 和 value 指定其内容。

在程序中要使用连接字符串时，可以通过静态类 ConfigurationManager 来获取，代码如下：

```
string connstr = ConfigurationManager.ConnectionStrings["connstr"].ConnectionString;
```

或

```
string connstr = ConfigurationManager.AppSettings["connstr"];
```

前者从<connectionStrings>节中获取连接字符串，后者从<appSettings>节中获取。由于 ConfigurationManager 类定义在 System.Configuration 命名空间中，所以代码开始要导入该命名空间，如下所示：

```
using System.Configuration;
```

9.2.3 使用 Command 对象

建立与数据源的连接后，可以使用 Command 对象来执行命令并从数据源中返回结果。每一种.NET 数据提供程序都有一个特定的 Command 对象，实际应用中应正确选择。适用于 OLE DB 驱动的是 OleDbCommand 对象，适用于 ODBC 驱动的是 OdbcCommand 对象，

适用于 SQL Server .NET 提供程序的是 SqlCommand 对象，适用于 Oracle .NET 提供程序的是 OracleCommand 对象。

1. 创建命令对象

可以使用各种命令对象的构造函数来创建命令对象。

例如：

```
// 创建命令对象,同时指定所用数据库连接及要执行的 SQL 命令
SqlCommand cmd = new SqlCommand( "select * from authors",  conn);
```

或

```
// 单独创建命令对象,然后通过属性指定所用连接和 SQL 命令
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "select * from authors";
cmd.Connection = conn;
```

也可以在连接对象上调用 CreateCommand()方法创建相关的命令对象，代码如下：

```
SqlCommand cmd = conn .CreateCommand( );
cmd.CommandText = "select * from authors";
```

2. 命令对象的常用成员

所有命令对象都从 System. Data. Common. DbCommand 类继承，其使用方法大致相同。它们都含有如下的常用属性：

- CommandText：获取或设置针对数据源运行的文本命令。该属性通常指定为一条 SQL 语句，或者一个存储过程的名称。
- CommandType：指定命令的类型，其取值为 System. Data. CommandType 枚举；若选 Text，表示要执行一条 SQL 语句；若选 StoredProcedure，表示要执行一个存储过程。
- Connection：获取或设置该命令对象所依赖的数据库连接对象。
- Parameters：获取命令参数的集合，当执行参数化查询时该属性非常重要。例如，根据书名查询一本书籍，就将书名作为参数传递进去。

命令对象还具有如下常用方法：

- Cancel()：尝试取消 DbCommand 的执行。
- Dispose()：释放命令对象所使用的所有资源。
- ExecuteNonQuery()：执行一个非查询的 SQL 语句或存储过程，如执行 Update 命令就需要调用该方法。
- ExecuteReader()：执行查询命令或存储过程，返回由 DataReader 封装的记录集。
- ExecuteScalar()：执行查询命令或存储过程，返回一个标量值(即单值)。该命令在聚合查询中非常有用，如要统计某一本书某天的销售量。

3 使用 ExecuteReader 方法

在命令对象上调用其 ExecuteReader 方法，可以执行一条 SELECT 语句，其返回值为 DataReader 对象。

例如：

```
SqlCommand cmd = conn .CreateCommand( );
cmd.CommandText = "select * from authors";
SqlDataReader dr = cmd.ExecuteReader();
```

在 Web 应用程序中,经常要根据用户的输入在数据库中查询满足条件的记录,这时需要动态构造一条 SQL 语句。通常采用字符串拼接的方式构造动态 SQL,请看如下示例。

例 9-3 在 NorthWind 数据库中,根据顾客号查询该顾客的订单信息,运行效果如图 9-7 所示。

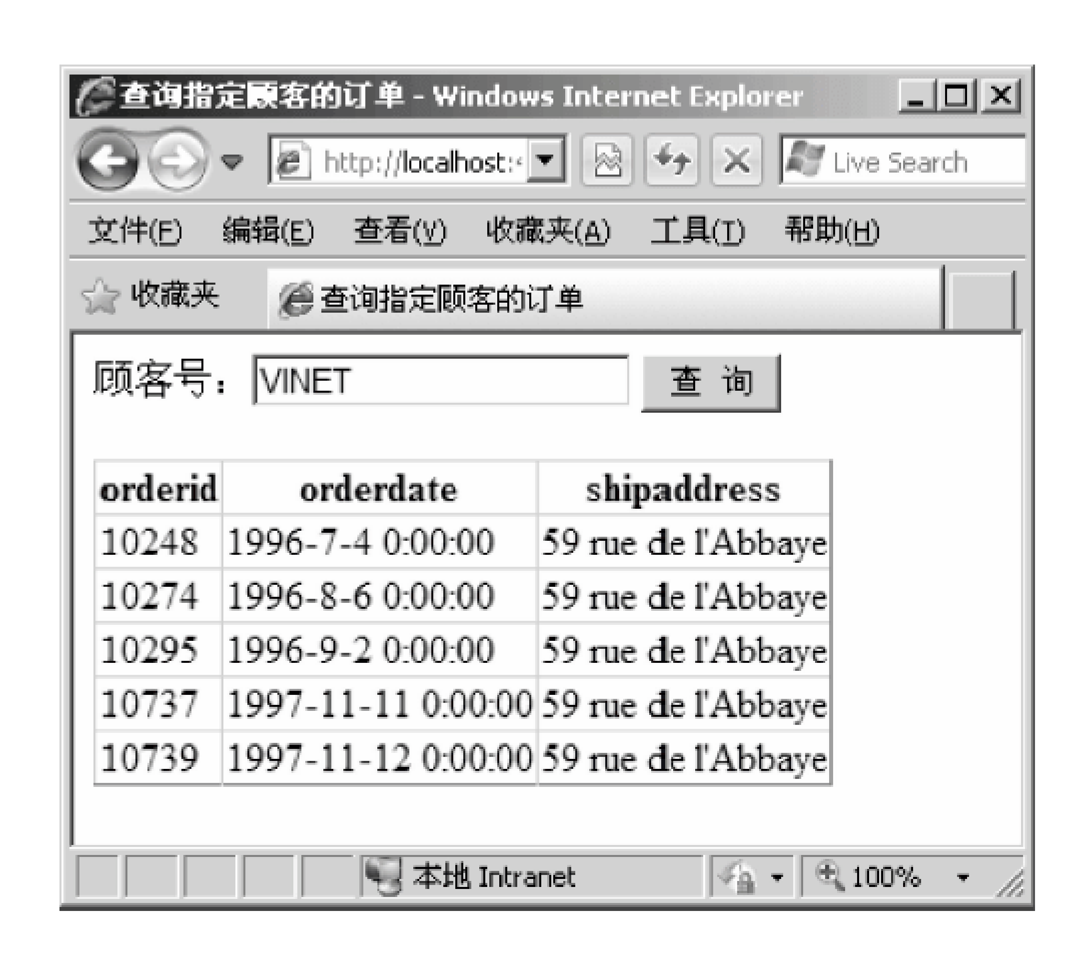


图 9-7 根据顾客号查询该顾客的订单页面

查询按钮的单击事件代码如下:

```
protected void btnqry_Click(object sender, EventArgs e)
{
    string connstr = ConfigurationManager.ConnectionStrings["connstr"].ConnectionString ;
    string sql = "select orderid, orderdate, shipaddress from orders where customerid = '" +
tbxuid.Text + "'";
    using (SqlConnection conn = new SqlConnection(connstr))
    {
        SqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = sql;
        conn.Open();
        SqlDataReader dr = cmd.ExecuteReader();
        GridView1.DataSource = dr;
        GridView1.DataBind();
        dr.Close();
    }
}
```

该示例中,查询用的 Select 语句采用字符串拼接的方式生成。当 Select 语句中查询条件较多时,可以使用 StringBuilder 对象辅助拼接。

例如:

```
StringBuilder sb = new StringBuilder();
sb.Append("select orderid, orderdate, shipaddress from orders where customerid = '");
sb.Append(tbxuid.Text);
sb.Append("'");
cmd.CommandText = sb.ToString();
```

使用字符串拼接的方式存在一定的安全隐患,容易被利用进行 SQL 注入攻击。为解决这个问题,通常使用参数化命令,即在 SQL 文本中使用占位符来代表命令参数,然后通过 Parameter 对象将参数值传递进来。例如,根据用户代码查询订单的程序中,用户代码可以作为参数输入,相关代码如下:

```
cmd.CommandText =
    "select orderid, orderdate, shipaddress from orders where customerid = @custid";
SqlParameter parameter = new SqlParameter("@custid", SqlDbType.VarChar);
parameter.Value = tbxuid.Text;
cmd.Parameters.Add(parameter);
SqlDataReader dr = cmd.ExecuteReader();
```

参数化命令中可以同时传递多个参数。例如,示例 9.3 中若要查询指定用户指定时间段内的所有订单,可以使用如下的命令格式:

```
cmd.CommandText = "select orderid, orderdate, shipaddress from orders where "
           + "customerid = @custid and orderdate between @fromdate and @todate";
cmd.Parameters.Add("@custid", SqlDbType.VarChar);
cmd.Parameters.Add("@startdate", SqlDbType.DateTime);
cmd.Parameters.Add("@enddate", SqlDbType.DateTime);
cmd.Parameters["@custid"].Value = tbxuid.Text;
cmd.Parameters["@startdate"].Value = "1996-01-01";
cmd.Parameters["@enddate"].Value = "1997-01-01";
```

参数占位符的语法取决于数据提供程序,如表 9-2 所示。

表 9-2　参数占位符的语法与数据提供程序的关系

数据提供程序	参数占位符语法
System.Data.SqlClient	以 @parametername 格式使用命名参数
System.Data.OracleClient	以 :parmname(或 parmname)格式使用命名参数
System.Data.OleDb	使用由问号(?)表示的位置参数标记
System.Data.Odbc	使用由问号(?)表示的位置参数标记

可以看出,SqlClient 和 OracleClient 提供程序都使用了命名参数,通过参数名明确区分各个参数,而 OleDb 和 Odbc 提供程序都没有为参数命名,只是用“?”通配符置换,当传递多个参数时,只能根据参数出现的先后顺序来进行区分,所以一定要保证参数的添加顺序与其在 SQL 字符串中的位置顺序相一致。

例如,使用 OleDb 方式连接 SQL Server 数据库,查询指定顾客指定时间段订单的示例代码如下:

```
using (OleDbConnection conn = new OleDbConnection(connstr))
{
    OleDbCommand cmd = conn.CreateCommand();
    cmd.CommandText = "select orderid, orderdate, shipaddress from orders "
                    + " where customerid = ? and orderdate between ? and ?";
    cmd.Parameters.AddWithValue("@custid", tbxuid.Text);
    cmd.Parameters.AddWithValue("@fromdate", "1996-01-01");
    cmd.Parameters.AddWithValue("@todate", "1997-01-01");
    conn.Open();
    OleDbDataReader dr = cmd.ExecuteReader();
    GridView1.DataSource = dr;
    GridView1.DataBind();
    dr.Close();
}
```

4. 使用 ExecuteScalar 方法

调用命令对象的 ExecuteScalar 方法将返回查询结果中第一行第一列的值，若结果中包含多行或多列，系统都将忽略其余数据。该方法常用于返回 SQL 聚合函数的查询结果。例如，要查询 Employees 表，统计并显示所有员工的数量，可使用如下代码：

```
using (SqlConnection conn = new SqlConnection(connstr))
{
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = "select count(*) from employees";
    conn.Open();
    int num = (int)cmd.ExecuteScalar();
    lblmsg.Text = string.Format("员工总数:<b>{0}</b>", num);
}
```

注意：ExecuteScalar 方法的返回结果为 object 类型，一定要根据命令结果强制转换成合适的数据类型。

5. 使用 ExecuteNonQuery 方法

调用命令对象的 ExecuteNonQuery 方法执行不返回结果集的命令，如 Insert、Update、Delete 等数据操纵语句，以及 Create Table 等数据定义语句。请看如下代码示例：

```
using (SqlConnection conn = new SqlConnection(connstr))
{
    conn.Open();
    SqlCommand cmd = conn.CreateCommand();
    // 执行插入操作
    cmd.CommandText = "Insert into employees(LastName, FirstName) values ('Tom', 'Cat')";
    int num =  cmd.ExecuteNonQuery();
    lblmsg.Text += string.Format("<br /> 共插入记录:<b> {0} 条</b>",num);
    // 执行更改操作
    cmd.CommandText = "Update employees set LastName = 'Jerry', FirstName = 'Mouse' "
                    + " where EmployeeID = 9";
    num = cmd.ExecuteNonQuery();
    lblmsg.Text += string.Format("<br /> 共修改记录:<b> {0} 条</b>",num);
    // 执行删除操作
```

```
        cmd.CommandText = "Delete from employees where EmployeeID > 9";
        num = cmd.ExecuteNonQuery();
        lblmsg.Text += string.Format("<br /> 共删除记录: <b> {0} 条</b>", num);
    }
```

ExecuteNonQuery 方法的返回值为整型数,代表执行命令后受影响的记录行数。

6. 执行存储过程

存储过程是保存在数据库中的可被批量执行的一条或多条 SQL 语句,与函数类似,具有良好的程序结构,可以通过输入参数接收数据,也可以通过输出参数返回数据。使用存储过程具有很多优点:

(1) 可以大幅度的提高程序性能。由于存储过程是多条语句的复合体,只访问一次数据库就可以完成很多工作,比起使用 Command 对象一次次的向数据库发送 SQL 语句,效率要高得多;另外,存储过程在数据库中进行了编译和优化,执行效率也提高了很多。

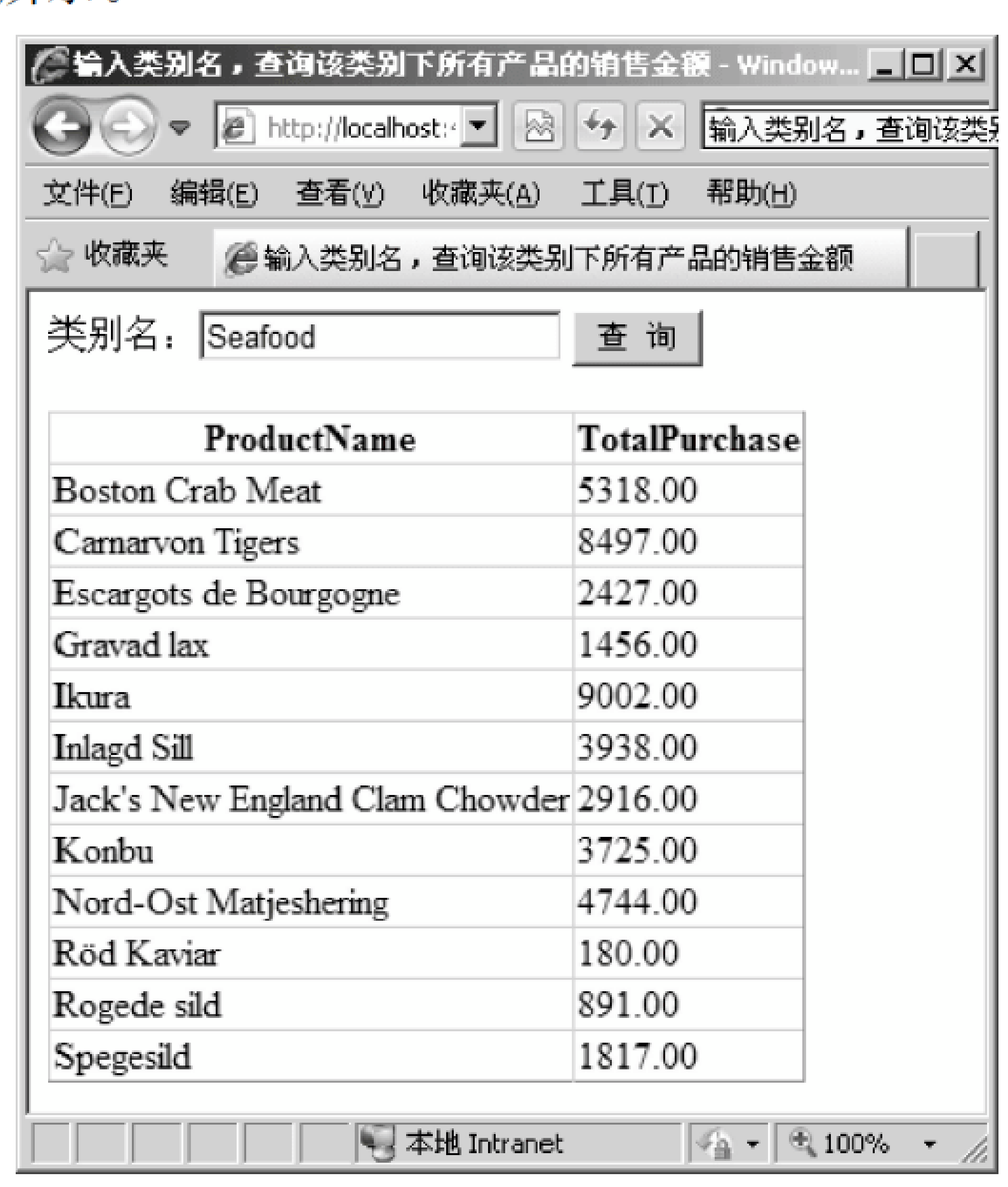
(2) 可以简化应用程序的设计。对于非常复杂的业务处理(如销售统计),若将处理逻辑封装在存储过程中,则应用程序中只需简单的调用存储过程就可以完成所有工作,有效地降低了程序的复杂度。

(3) 使程序更易于维护。将复杂的数据处理逻辑从应用程序中分离出来,可以对存储过程进行优化,只要接口不变,就不需要更改或重新编译应用程序。

(4) 有利于人员分工。大型项目中明确的人员分工异常重要,使用存储过程,可以将复杂的数据处理逻辑分工给数据库开发人员,任务更加明确。

使用 Command 对象调用存储过程的方式与执行普通 SQL 命令的方式类似,只是要将其 CommandType 属性设定为 StoredProcedure。

例 9-4 在 NorthWind 数据库中调用存储过程,查询指定类别下所有产品的销售量。程序运行效果如图 9-8 所示。



ProductName	TotalPurchase
Boston Crab Meat	5318.00
Carnarvon Tigers	8497.00
Escargots de Bourgogne	2427.00
Gravad lax	1456.00
Ikura	9002.00
Inlagd Sill	3938.00
Jack's New England Clam Chowder	2916.00
Konbu	3725.00
Nord-Ost Matjeshering	4744.00
Röd Kaviar	180.00
Rogede sild	891.00
Spegesild	1817.00

图 9-8　根据类别查询产品销售金额程序的运行效果

打开 NorthWind 数据库，在“可编程性/存储过程”菜单下可看到系统内置的几个存储过程，其中就有 SalesByCategory，本例将调用该存储过程。

建立 Web 窗体，为查询按钮的单击事件编写如下代码：

```
protected void btnqry_Click(object sender, EventArgs e)
{
    string connstr =
    ConfigurationManager.ConnectionStrings["nwconnstr"].ConnectionString;
    using (SqlConnection conn = new SqlConnection(connstr))
    {
        SqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = "SalesByCategory";
        cmd.CommandType = CommandType.StoredProcedure;
        SqlParameter parameter = new SqlParameter("@CategoryName",
            SqlDbType.VarChar);
        parameter.Value = tbxcatname.Text;
        cmd.Parameters.Add(parameter);
        conn.Open();
        SqlDataReader dr = cmd.ExecuteReader();
        GridView1.DataSource = dr;
        GridView1.DataBind();
        dr.Close();
    }
}
```

本例中，将命令对象的 CommandText 属性设置为存储过程的名称，CommandType 属性设定为命令类型枚举中的 StoredProcedure，然后通过参数将商品类别传递给存储过程。由于该过程执行后将返回记录集，所以调用了命令对象的 ExecuteReader 方法。

若存储过程的返回结果不是记录集，一般通过命令对象的 ExecuteNonQuery 方法调用。

例 9-5 向 Employee 表中插入一条记录，并返回该员工的 ID 号。

在 Employee 表中，EmployeeID 为标识字段，由系统字段生成。若使用 Command 对象执行一条 Insert 语句来插入员工，那么要得到刚插入记录的员工号将是一件很麻烦的事情。使用存储过程可以解决这个问题，在存储过程中执行完 Insert 语句后，立刻调用@@IDENTITY 函数即可得到新添加记录的标识。

在数据库执行如下脚本，建立 InsertEmployee 存储过程：

```
CREATE PROCEDURE InsertEmployee
    @LastName varchar(20),
    @FirstName varchar(10),
  @EmployeeID int output
AS
  Insert into Employees (LastName, FirstName) Values (@LastName, @FirstName);
  set @EmployeeID = @@IDENTITY;
```

该存储过程有三个参数，前两个为输入参数，用于接收员工的名和姓，最后一个为输出

参数,可以将新添加记录的员工号带回来。

下一步是在应用程序中调用该存储过程。建立添加员工记录的 Web 页面,在“添加”按钮的单击事件中编写如下代码:

```
string connstr =
ConfigurationManager.ConnectionStrings["nwconnstr"].ConnectionString;
using (SqlConnection conn = new SqlConnection(connstr))
{
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = "InsertEmployee";
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Parameters.Add("@LastName", SqlDbType.VarChar, 20);
    cmd.Parameters.Add("@FirstName", SqlDbType.VarChar, 10);
    cmd.Parameters["@LastName"].Value = tbxlname.Text;
    cmd.Parameters["@FirstName"].Value = tbxfname.Text;
    cmd.Parameters.Add("@EmployeeID", SqlDbType.Int);
    cmd.Parameters["@EmployeeID"].Direction = ParameterDirection.Output;
    conn.Open();
    int n = cmd.ExecuteNonQuery();
    lblmsg.Text += string.Format("Inserted {0} Records <br />", n);
    int empid = (int)cmd.Parameters["@EmployeeID"].Value;
    lblmsg.Text += "New ID : " + empid.ToString();
}
```

上述程序为命令对象 cmd 添加了三个参数:@LastName、@FirstName 和 @EmployeeID。前两个参数没有指定 Direction 属性,默认为 Input,即输入参数;第三个参数要从存储过程中带回一个值,所以定义为输出参数,将其 Direction 属性设置为 Output。由于该存储过程不返回记录集,所以调用命令对象的 ExecuteNonQuery 方法;该方法实际上带回了两个值,一是方法的返回值,代表 Insert 语句执行后影响的记录条数,本例为 1;二是新添加记录的标识号,通过输出参数获得,即:

```
int empid = (int)cmd.Parameters["@EmployeeID"].Value;
```

该程序的运行效果如图 9-9 所示。



图 9-9 插入员工页面的运行效果

9.2.4 使用 DataReader 对象

数据库系统中最常用的操作是数据检索，使用 Select 语句完成，检索的结果是一个记录集。在应用程序中通常有两种方式访问该记录集：一是使用 DataReader；二是使用 DataTable。

DataReader 允许以只读、只进的方式，每次读取一条记录进行处理。该方式占用内存资源极小，操作效率极高，是获取数据最简单高效的方式。

每种 .NET 数据提供程序都定义了各自的 DataReader 类，有 SqlDataReader、OracleDataReader、OleDbDataReader、OdbcDataReader，都从 DbDataReader 类继承，其核心成员如表 9-3 所示。

表 9-3　DataReader 对象的核心成员

成　　员	描　　述
HasRows 属性	指示该 DataReader 中是否包含数据
FieldCount 属性	获取当前行中的列数
Read 方法	将游标移动到记录集的下一行
GetValue 方法	获取当前行中指定序号的字段的值，系统将根据数据源中该字段的数据类型匹配一个最相近的.NET 数据类型返回
GetXxx 方法	获取当前行中指定序号的字段的值，但明确指定了返回值的类型，所以返回类型与方法名称中指定的类型一致。如 GetInt32 ( )、GetChar ( )、GetDateTime( )等
Close 方法	关闭 DataReader 对象

使用 DataReader 对象操作记录集时，要先移动游标定位到指定行，然后再获取各列的数据。刚得到 DataReader 时，游标位于第一条记录的前面，这时还无法读取数据，必须先调用 Read 方法，让游标下移一行，然后才可以操作。Read 方法返回一个布尔值，表示游标指向的位置是否有数据，只有返回 true 才可以读取数据，若返回 false 则说明已经没有数据。

例 9-6　查询并显示 Categories 表中所有类别的 ID 和名称。

建立 Web 页面，为 Page_Load 事件编写如下代码：

```
string connstr = ConfigurationManager.ConnectionStrings["nwconnstr"].ConnectionString;
using (SqlConnection conn = new SqlConnection(connstr))
{
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = "Select CategoryID, CategoryName from Categories";
    conn.Open();
    SqlDataReader reader = cmd.ExecuteReader();
    StringBuilder sb = new StringBuilder("");
    sb.Append("<table><tr><td>CategoryID</td><td>CategoryName</td></tr>");
    while (reader.Read())
    {
        sb.Append("<tr><td>");
        sb.Append(reader[0]);
```

```
            sb.Append("</td><td>");
            sb.Append( reader["CategoryName"]);
            sb.Append("</td></tr>");
        }
        sb.Append("</table>");
        reader.Close();
        lblmsg.Text = sb.ToString();
    }
```

可以看出,得到 DataReader 对象后,在 while 循环中不断调用其 Read 方法遍历记录集,直到游标指向记录集末尾。游标每定位到某一行,系统会自动将该行数据获取到内存中,这时向 DataReader 传递列的名称或序号,就可以访问指定列的数据。

使用 GetValue 方法或 GetXxx 方法,传递列号,也可以获取指定列的数据。使用后一种方法,明确的指定了所获取的数据的类型,执行效率更高。

9.3 使用 DataSet 架构

DataReader 是基于连接的对象,只有在数据处理完成后才可以断开与数据源的连接。DataSet 架构则使用了 ADO.NET 非连接的特性,可以将批量数据读入内存,然后进行离线的处理,最后可以将处理结果批量写回数据库中。为实现 DataSet 与数据库的交互,通常使用 DataAdapter 对象作为桥梁。

DataSet 中包含两种核心元素,一是表的集合;二是表间关系的集合。前者代表数据,后者代表约束。

9.3.1 使用 DataTable

DataTable 是 DataSet 架构的核心,代表内存中的表。依靠它可以离线处理数据,可以前、后移动游标定位记录,快速检索记录,对记录进行排序,按条件过滤记录等。DataTable 既可以包含在 DataSet 中,也可以游离于 DataSet 之外而独立存在(这时将无法建立表间关系,也无法实现完整性约束)。

1. 创建 DataTable 对象

DataTable 中主要包含两方面信息:一是架构,通过列的集合指定;二是数据,通过行的集合指定。

若以编程的方式创建 DataTable,则要先创建各列的 DataColumn 对象并将其添加到 DataColumnCollection 中,这样才能确定表格的架构。请看如下示例:

```
DataTable dt = new DataTable();
DataColumn col = dt.Columns.Add("EmployeeID", typeof(Int32));
col.Unique = true;
dt.Columns.Add("LastName", typeof(String));
dt.Columns.Add("FirstName", typeof(String));
```

通过 DataTable 对象的 Columns 属性可以获取其列集,在列集上调用 Add 方法,并传

入列名和数据类型来创建列对象，还可以在列对象上指定约束。例如，本例中对 CustID 列设定了唯一约束，即所有记录的 CustID 不能重复。

2. 向 DataTable 中添加数据

DataTable 的数据包含在行集中，每行数据用一个 DataRow 对象表示。可通过编程方式向 DataTable 的行集中添加行数据，示例代码如下：

```
DataRow row = dt.NewRow();
row[0] = 1;
row["LastName"] = "Tom";
row["FirstName"] = "Cat";
dt.Rows.Add(row);
```

调用 DataTable 对象的 NewRow 方法返回一个新行，行中包含若干个列，通过指定列的序号或名称可以访问指定列，本例中通过这种方式向三个列分别赋值。通过 DataTable 对象的 Rows 属性可以访问其行集，在行集上调用 Add 方法，可以将行对象添加到表中，表中于是有了数据。

通常都是从数据库中检索数据并填充到 DataTable 中，最直接的方式是调用 DataTable 对象的 Load 方法，并传入一个 DataReader 对象，这样系统会自动从 DataReader 中不断获取数据并装载到 DataTable 中。请看如下示例：

```
cmd.CommandText = "Select EmployeeID, LastName, FirstName from Employees";
SqlDataReader reader = cmd.ExecuteReader();
dt.Load(reader);
```

在 DataSet 架构中通常使用 DataAdapter 对象来创建数据表并向其中填充数据，这些内容将在后续章节中介绍。

当使用 Load 方法装载数据，或使用 DataAdapter 对象填充数据时，由于可以连接到数据库，自动获取表的架构信息，所以不需要专门编写代码定义数据表结构，只要创建空的 DataTable 对象，即可开始装载或填充数据。

3. 遍历表中数据

DataTable 的 Rows 属性返回行的集合，其中每个元素就是一条记录，通常使用 Foreach 循环遍历行集中的每一个 DataRow。DataRow 又是字段值的容器，可以通过字段序号或名称访问它们。下面的代码演示了如何遍历并显示 DataTable 中的数据。

```
StringBuilder sb = new StringBuilder("<table>");
sb.Append("<tr><td>FirstName</td><td>LastName</td></tr>");
foreach (DataRow row in dt.Rows)
{
    sb.Append("<tr><td>");
    sb.Append(row["FirstName"].ToString());
    sb.Append("</td><td>");
    sb.Append(row["LastName"].ToString());
    sb.Append("</td></tr>");
}
sb.Append("</table>");
```

```
lblmsg.Text = sb.ToString();
```

4. 检索表中数据

DataTable提供了一个有用的方法Select,可以返回满足条件的行集;该方法使用的表达式与SQL Select语句中Where子句的作用类似,只是Select方法在内存表中查询,不连接数据库。例如,假设productdt数据表中包含着所有商品的信息,要查找类别号为2的商品并显示其名称,可使用如下的代码片段。

```
DataRow[ ] matchrows = dt.Select("CategoryID = 2");
StringBuilder sb = new StringBuilder("<ul>");
foreach(DataRow row in matchrows)
{
    sb.Append("<li>");
    sb.Append(row["ProductName"].ToString());
    sb.Append("</li>");
}
sb.Append("</ul>");
lblmsg.Text = sb.ToString();
```

由于Select方法的返回值为行集,所以使用了DataRow的数组来保存查询结果,然后使用foreach循环遍历所有行。

DataTable中也允许按主键检索特定行,这要先在DataTable上定义主键,然后在行集上使用Find方法。请看如下代码片段:

```
// 为数据表定义主键列
DataColumn[] columns = new DataColumn[1];
columns[0] = dt.Columns["ProductID"];
dt.PrimaryKey = columns;
// 根据主键进行检索(查询商品号为5的行)
DataRow findrow = dt.Rows.Find(5);
// 输出检索结果
if (findrow != null)   lblmsg.Text = findrow["ProductName"].ToString();
```

9.3.2 使用DataView

DataView为DataTable对象定义数据视图,使DataTable能够支持自定义过滤和数据排序。在数据绑定的场合中,利用DataView可以只选出表中的一部分数据进行显示,也可以按不同的方式排序显示数据;而在实现这些功能的同时,不会影响DataTable中的真实数据。

每个DataTable都有一个默认的DataView与之关联,使用DataTable对象的DefaultView属性可以引用该数据视图;也可以在同一个表上创建多个表示不同视图的DataView对象。

1. 数据排序

借助DataView对象的Sort属性,设置合适的排序表达式,即可实现数据排序。下面的

示例演示了如何对 Products 表中的数据进行排序显示。

```
DataTable dt = new DataTable();
string connstr =
ConfigurationManager.ConnectionStrings["nwconnstr"].ConnectionString;
using (SqlConnection conn = new SqlConnection(connstr))
{
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = "Select ProductID, ProductName,UnitPrice from Products ";
    conn.Open();
    SqlDataReader reader = cmd.ExecuteReader();
    dt.Load(reader);
}
GridView1.DataSource = dt.DefaultView;
GridView1.DataBind();
DataView dv = new DataView(dt);
dv.Sort = "UnitPrice";
GridView2.DataSource = dv;
GridView2.DataBind();
```

程序的运行结果如图 9-10 所示。示例中第一个网格(GridView)使用了默认数据视图，第二个网格使用了按 UnitPrice 列排序的视图。排序表达式中可以使用 ASC 或 DESC 指定按升序或降序排列，若要按多个字段组合排列，可以使用如下的代码形式：

```
dv.Sort  =  " ProductName , UnitPrice ";
```



原始数据：

ProductID	ProductName	UnitPrice
7	Uncle Bob's Organic Dried Pears	30.0000
14	Tofu	23.2500
28	Rössle Sauerkraut	45.6000
51	Manjimup Dried Apples	53.0000
74	Longlife Tofu	10.0000

按名称排序：

ProductID	ProductName	UnitPrice
74	Longlife Tofu	10.0000
14	Tofu	23.2500
7	Uncle Bob's Organic Dried Pears	30.0000
28	Rössle Sauerkraut	45.6000
51	Manjimup Dried Apples	53.0000

图 9-10　使用 DataView 进行数据排序

2. 数据过滤

借助 DataView 对象的 RowFilter 属性,可以自定义过滤条件,将 DataTable 中满足条件的记录选择出来显示。RowFilter 属性和 SQL 查询中的 Where 子句功能类似,条件表达式的书写格式也基本相同,详细用法请参阅 MSDN 文档。下面的代码片段演示了如何从 DataTable 中选出价格超过 50 元的记录及产品名以 M 开头的记录分别显示。

```
dt.DefaultView.RowFilter = "UnitPrice > 40";
GridView1.DataSource = dt.DefaultView;
GridView1.DataBind();
DataView dv = new DataView(dt);
dv.RowFilter = "ProductName Like 'M%'";
GridView2.DataSource = dv;
GridView2.DataBind();
```

9.3.3 使用 DataRelation

在 DataSet 中不但可以定义数据表,还可以定义表之间的关系,以方便的实现数据导航。例如,在客户表和订单表之间建立 DataRelation 后,可以方便的根据客户号检索该客户的所有订单。请看如下的代码片段:

```
// 在 Customers 表和 Orders 表之间建立关系
DataRelation customerOrdersRelation =
    customerOrders.Relations.Add("CustOrders",
    customerOrders.Tables["Customers"].Columns["CustomerID"],
    customerOrders.Tables["Orders"].Columns["CustomerID"]);
// 遍历 Customers 表中的每一个顾客
foreach (DataRow custRow in customerOrders.Tables["Customers"].Rows)
{
    Console.WriteLine(custRow["CustomerID"].ToString());
    // 根据 DataRelation 获取该顾客的所有订单
    foreach (DataRow orderRow in custRow.GetChildRows(customerOrdersRelation))
    {
        Console.WriteLine(orderRow["OrderID"].ToString());
    }
}
```

其中,customerOrders 为 DataSet 对象,在它的关系集中添加了一个新的关系,这样,在遍历主表数据时(外层循环),可以方便地按照 CustomerID 找到子表该顾客的所有订单,并遍历显示。

1. 建立数据表之间的关系

使用 DataRelation 使两个 DataTable 通过 DataColumn 对象彼此关联,类似于数据库中的"主键/外键"关系。DataRelation 是在父表和子表中的匹配列之间创建的,要求这两个列的数据类型必须相同,但列名可以不同。

建立数据表间关系的代码如下:

```
// 在主表和子表中确定建立关系的匹配列
DataColumn parentColumn = DataSet1.Tables["Customers"].Columns["CustID"];
```

```
DataColumn childColumn = DataSet1.Tables["Orders"].Columns["CustID"];
// 根据匹配列建立关系
DataRelation relCustOrder = new DataRelation (
                        "CustomersOrders",  parentColumn,  childColumn );
```

调用 DataRelation 类的构造器时需要传入三个参数：关系的名称、主表中的列（相当于主键）及子表中的列（相当于外键）。

关系创建后，还要将其添加到数据集的关系集合中，代码如下：

```
DataSet1.Relations.Add(relCustOrder);
```

通过 DataSet 对象的 Relations 属性得到关系集合，再调用其 Add 方法将新创建的关系加入集合；也可以通过实例中的方法，创建关系的同时将其加入集合中。

2. 根据关系进行数据导航

建立关系的主要目的是在主表和子表之间进行数据导航，即根据主表中某行的数据，在子表中获取与之匹配的一系列行，或根据子表中的某行数据，获取主表中的相应行。

主表的行对象调用 GetChildRows 方法，可得到子表中与之匹配的行。

例如：

```
DataRow [] orderRows = customerRow. GetChildRows(customerOrdersRelation);
```

子表的行对象调用 GetParentRow 方法，可获得主表中的匹配行。

例如：

```
DataRow customerRow = orderRow. GetParentRow (customerOrdersRelation);
```

9.3.4 使用 DataAdapter

在 ADO.NET 体系中，DataAdapter 作为数据源与 DataSet 之间的桥梁，起着承上启下的作用，可以将数据源中的数据填充到 DataSet，也可以将 DataSet 中所作的数据更改保存到数据源中。

DataAdapter 建立在 Connection 之上，内部包含 4 个命令对象：SelectCommand、InsertCommand、UpdateCommand 和 DeleteCommand。SelectCommand 对象用于从数据源中检索数据，其他三个对象用于将数据更改写回数据源。

1. 填充数据

DataAdapter 的 4 个命令对象中，只有 SelectCommand 是必需的。当调用 DataAdapter 的 Fill 方法时，实际上是先使用 SelectCommand 检索数据，然后再填充到 DataTable 中。请看如下的示例代码：

```
string connstr = ConfigurationManager.ConnectionStrings["nwconnstr"].ConnectionString;
SqlConnection conn = new SqlConnection(connstr);
string sql = "Select EmployeeID, LastName, FirstName from Employees";
// 创建 DataAdapter 对象,需传入要执行的查询语句及连接对象
```

```
SqlDataAdapter da = new SqlDataAdapter(sql, conn);
// 创建 DataSet 对象,并向其中的 employees 表中填充数据
DataSet ds = new DataSet();
da.Fill(ds, "employees");
StringBuilder sb = new StringBuilder("<table>");
sb.Append("<tr><td>FirstName</td><td>LastName</td></tr>");
// 遍历包含在 DataSet 中的表中的数据
foreach (DataRow r in ds.Tables["employees"].Rows)
{
    sb.Append("<tr><td>");
    sb.Append(r["FirstName"].ToString());
    sb.Append("</td><td>");
    sb.Append(r["LastName"].ToString());
    sb.Append("</td></tr>");
}
sb.Append("</table>");
lblmsg.Text = sb.ToString();
```

创建 DataAdapter 对象时,需要指定其连接属性及查询语句,以便构建 SelectCommand 对象;也可以先创建好 Command 对象,然后将它赋给 DataAdapter 对象的 SelectCommand 属性。调用 Fill 方法向数据表中填充数据,第一个参数指定 DataSet 对象,第二个参数指定要填充的 DataTable 名称;若 DataSet 中还没有该名称的数据表,则自动创建。

注意:这段代码中并没有打开和关闭数据库连接的代码,这些工作是由 DataAdapter 自动执行的。当调用 Fill 方法时,DataAdapter 以隐含的方式,先打开到数据源的连接,再执行 Select 命令,最后关闭连接。

2. 更新数据

调用 DataAdapter 的 Update 方法可以将 DataSet 中的更改写回数据源。Update 方法将 DataSet 的实例和可选的 DataTable 对象或 DataTable 名称用作参数。如果未指定 DataTable,则使用 DataSet 中的第一个 DataTable。

当调用 Update 方法时,DataAdapter 会分析已做的更改;当遇到对 DataRow 的更改时,将使用 InsertCommand、UpdateCommand 或 DeleteCommand 来处理该更改。在调用 Update 之前,必须先显式的设置这些命令;如果调用了 Update 但不存在用于特定更新的命令则会引发异常。请看如下示例:

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    // 创建 DataAdapter 对象并设置其 SelectCommand,以便填充数据
    SqlDataAdapter dataAdpater = new SqlDataAdapter(
        "SELECT CategoryID, CategoryName FROM Categories", connection);
    // 创建 UpdateCommand 对象,以便更新数据
    dataAdpater.UpdateCommand = new SqlCommand(
        "UPDATE Categories SET CategoryName = @CategoryName " +
        "WHERE CategoryID = @CategoryID", connection);
    dataAdpater.UpdateCommand.Parameters.Add(
```

```
            "@CategoryName", SqlDbType.NVarChar, 15, "CategoryName");
        dataAdpater.UpdateCommand.Parameters.Add(
            "@CategoryID", SqlDbType.Int);
        // 检索并填充数据
        DataTable categoryTable = new DataTable();
        dataAdpater.Fill(categoryTable);
        // 更新数据集中的数据
        DataRow categoryRow = categoryTable.Rows[0];
        categoryRow["CategoryName"] = "New Beverages";
        // 将数据集中的更新保存到数据源
        dataAdpater.Update(categoryTable);
    }
```

3. 使用 DbCommandBuilder 对象辅助更新数据

为方便执行数据更新操作，通常使用 CommandBuilder 对象辅助 DataAdapter。

DbDataAdapter 不会自动生成更新数据源的 SQL 语句，但只要设置了 SelectCommand 属性，就可以为其创建 DbCommandBuilder 对象来自动生成 SQL 语句进行单表更新。

例 9-7 创建 DbCommandBuilder 对象来自动生成 SQL 语句进行单表更新。

```
SqlDataAdapter adapter = new SqlDataAdapter();
adapter.SelectCommand = new SqlCommand(sql, connection);
SqlCommandBuilder builder = new SqlCommandBuilder(adapter);
connection.Open();

DataSet dataSet = new DataSet();
adapter.Fill(dataSet, tableName);
// 这里可以写更新 DataSet 的代码
builder.GetUpdateCommand();            // 自动构造更新数据库的 SQL
adapter.Update(dataSet, tableName);    // 将更改写回数据库
```

关于 CommandBuilder 对象的使用细节请参阅 MSDN 文档，这里不再说明。

9.4 习题和上机练习

1. 简答题

(1) 列举 ADO.NET 体系结构中的常用对象，并说明各对象的作用。

(2) 试比较 DataReader 与 Dataset 对象的异同，并思考什么情况下使用 DataReader，什么情况下使用 Dataset。

(3) 什么叫做 SQL 注入？如何预防 SQL 注入？请举例说明。

(4) 什么是存储过程？使用存储过程有什么好处？

(5) 在 ADO.NET 中调用存储过程与执行 SQL 命令的方法有什么区别？

(6) 如何在 Web.config 文件中保存连接字符串，如何在程序中访问该字符串？

(7) 若要连接 SQL Server 2005 数据库，试举例说明如何配置可信的数据库连接。若要创建基于用户名和密码的数据库连接，试说明如何配置连接字符串，并说明在 SQL Server 中还要如何设置。

(8) ADO.NET 中,如何保证多次使用的数据库连接是来自同一个连接池?

(9) 在使用 Command 对象操作数据库时,什么情况下应调用其 ExecuteReader 方法?什么情况下应调用 ExecuteScalar 方法?什么情况下应调用 ExecuteNonQuery 方法?

(10) 试举例说明如何遍历 DataTable 中的数据。

(11) 试举例说明如何在 DataTable 中按主键快速检索特定行的数据。

(12) 什么是 DataView?试举例说明如何使用 DataView 实现数据排序和过滤。

2. 上机练习

新建一个数据库 test,其中建一个用户信息表 userinfo,包括用户名、密码、身份、姓名、性别、生日(可选用日期型)、电话、邮箱等信息,至少要有一个管理员身份的用户。

(1) 建立一个注册页面。

在注册页面,单击"注册"按钮后,用户输入的注册信息格式化显示(可以使用 CSS 文件),然后单击"确认"按钮,将信息写入数据表 userinfo 中。如此写入至少 5 条数据。

(2) 建立一个登录页面。

在登录页面,使用数据表中输入的用户名和密码后,单击"登录"按钮,进行身份验证,正确则进入下一个页面。

第 10 章　数 据 绑 定

几乎所有的 Web 应用程序都要和数据打交道，特别是还需要一种方便灵活的方式将数据展示在页面上，这就需要数据绑定技术。ASP.NET 提供了一个全能的数据绑定模型，允许将单个数据或数据集合绑定到特定控件上，由控件负责数据的展示。这样，就不需要编写复杂的代码，循环读取记录和字段来生成展示页面。如果借助 ASP.NET 提供的数据源控件，可以在页面和数据源之间定义一个声明性的连接，甚至不用写一行代码，就可以配置出一个具有数据库增、删、改、查功能的复杂页面。

10.1　数据绑定基础

数据绑定就是把数据源和控件相关联，由控件负责自动显示数据的一种方式。

ASP.NET 中的大部分控件（如 Label、TextBox、Image 等）都支持单值数据绑定，可以将控件的某个属性绑定到数据源，进而自动获取数据源的值；还有很多控件支持重复值绑定，也就是说它们可以呈现出一组项目（以列表或表格的方式），可以绑定到一个数据集合（如 DataReader 或 DataTable）上，自动、重复地获取集合中的每一项并呈现在页面上。

10.1.1　数据绑定表达式

在 ASP.NET 页面中使用数据绑定表达式可以输出页面的属性值、成员变量值或函数的返回值，前提是这些属性、成员变量及函数具有受保护的（Protected）或者公有的（Public）可见性。数据绑定表达式的一般格式如下：

```
<%# data_bind_expression %>
```

例如，假设页面中定义了一个叫 EmployeeName 的公共的或受保护的变量，则使用以下的表达式可以在页面上输出该变量的值。

```
<%# EmployeeName %>
```

还可以使用在运行时可计算的表达式来构造数据绑定表达式。

例如：

```
<%# getUserName( ) %>
<%# "Tom" + "Cat" %>
<%# DateTime.Now %>
<%# Request.Url %>
```

上面第一行代码调用了 getUserName 方法，第二行计算字符串表达式的值并输出，第三行获取当前时间并显示，第四行获取当前页面的 URL 并显示。

10.1.2 单值绑定

几乎可以将数据绑定表达式放置在页面的任何地方，但通常的做法是将其赋值给控件的某个属性。

例如：

```
<asp:Label ID = "lblUser" runat = "server" Text = "<% # CurrentUser %>"></asp:Label>
```

为显示数据绑定表达式的值，必须要在页面或控件上调用其 DataBind 方法，这时 ASP.NET 才检查页面上的表达式并用适当的值替换它们，若忘记了调用 DataBind 方法，数据绑定表达式将不会被填入值，在页面呈现时将被丢弃。

例 10-1 使用数据绑定表达式实现单值绑定。

首先建立如下的测试页面：

```
<html xmlns = "http://www.w3.org/1999/xhtml" >
<head runat = "server"><title>测试数据绑定表达式</title></head>
<body>
<form id = "form1" runat = "server">
  当前时间：<% # DateTime.Now %><br />
  当前页面：<% # Request.Url %><br />
  欢迎你：<asp:Label ID = "lblUser" runat = "server" Text = "<% # CurrentUser %>"></asp:
Label>
  <asp:Image ID = "imgUser" runat = "server" ImageUrl = "<% # getImg() %>" />
</form>
</body>
</html>
```

该页面中调用了 CurrentUser 属性和 getImg 方法，需要在后台代码中做如下定义：

```
protected string CurrentUser
{
    get
    {
        return "White";
    }
}
protected string getImg()
{
    return "./img/user.png";
}
```

最后需要注意，为计算并显示数据表达式的值，通常在页面的 Page_Load 事件中调用 Page 对象的 DataBind()方法，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    this.DataBind();
}
```

该页面的显示效果如图 10-1 所示。



图 10-1 单值数据绑定示例的显示效果

10.1.3 重复值绑定

重复值绑定允许将一个列表的信息绑定到一个控件上，列表可以是自定义对象的集合（如 ArrayList 或 Hashtable 等），也可以是行的集合（如 DataReader 或 DataTable 等）。

ASP.NET 提供了几个支持重复值绑定的基本列表控件，如 DropDownList、ListBox、CheckBoxList、RadioButtonList、BulletedList 等，它们具有如表 10-1 所示的基本属性。

表 10-1 列表控件的基本属性

属 性 名	属 性 描 述
DataSource	数据源对象，包含要显示的数据，该对象通常实现 ICollection 接口
DataSourceID	数据源对象的 ID，通过该属性可以链接列表控件和数据源控件。该属性与 DataSource 属性只能设置一个，不能同时使用
DataTextField	数据源中可以包含多个数据项（列），但列表控件中只能显示单个列的值，DataTextField 属性指定要显示在页面上的字段的名称（绑定到行集时）或属性名称（绑定到对象集时）
DataValueField	该属性和 DataTextField 属性类似，但从数据项中获得的数据不会显示在页面上，而是保存在底层 HTML 标签的 value 属性上，允许以后在代码中读取该属性值。该属性通常用于保存唯一值或主键
DataTextFormatString	定义一个可选的字符串，用于格式化 DataTextField 的值

为将列表的值绑定到列表控件上，通常有两种做法：

(1) 在代码中进行数据绑定：设置列表控件的 DataSource 属性为集合对象，然后显式地调用列表控件的 DataBind 方法实现数据绑定。

（2）使用声明的方式绑定：设置列表控件的 DataSourceID 属性为集合对象，不需要在代码中调用 DataBind 方法，系统就会自动执行数据绑定。

下面是一个在代码中进行数据绑定的例子，声明式数据绑定通常要配合数据源控件使用，将在后续章节举例。

例 10-2 建立 NorthWind 数据库的按类别查询产品信息的页面。

运行效果如图 10-2 所示。



图 10-2 按类别查询产品信息的页面

页面中使用下列列表框显示并选择产品类别，查询该类别的产品，并使用 BulletedList 控件显示所有产品的名称。

页面加载事件的代码如下，注意其中的加粗部分：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        using (SqlConnection conn = new SqlConnection(connstr))
        {
            SqlCommand cmd = conn.CreateCommand();
            cmd.CommandText = "Select CategoryID, CategoryName from Categories";
            conn.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            ddlcategory.DataSource = reader;
            ddlcategory.DataTextField = "CategoryName";
            ddlcategory.DataValueField = "CategoryID";
            ddlcategory.DataBind();
        }
        bindproducts();
    }
}
```

为将 DataReader 中的数据绑定到下列列表框，需要设置列表框的 DataSource 属性为 DataReader 对象；在列表框中要显示的是类别名，而当选择项改变时要提交给服务器的却是类别号，所以应设置列表框的 DataTextField 属性为 CategoryName，DataValueField 属性

为 Categoryid；最后，显式调用列表框的 DataBind 方法实现数据绑定。

根据所选类别查询产品的事件过程代码如下：

```
private void bindproducts()
{
    string catid = ddlcategory.SelectedValue;
    using (SqlConnection conn = new SqlConnection(connstr))
    {
        SqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = "Select ProductName from Products where CategoryID = @catid";
        cmd.Parameters.AddWithValue("@catid", catid);
        conn.Open();
        SqlDataReader reader = cmd.ExecuteReader();
        bllproduct.DataSource = reader;
        bllproduct.DataTextField = "productname";
        bllproduct.DataBind();
    }
}
```

这里，将查询到的产品名称绑定到 BulletedList 对象上，需要设置 BulletedList 对象的 DataSource 属性及 DataTextField，然后调用 DataBind 方法。

10.2 数据源控件

前面的章节中，介绍了通过代码访问数据库的方式、连接数据库、执行查询及更新操作，以及循环遍历记录集并将数据显示在页面上。为大幅度提高开发效率，减少编码量，还可以使用数据源控件，配合功能强大的数据绑定控件，甚至不用编写一行代码，就可以开发出基本的数据访问程序。

10.2.1 数据源控件概述

例 10-3 使用数据源控件和数据绑定控件，开发一个员工管理的应用程序，实现员工信息的增、删、改、查功能以及分页、排序显示功能。

请按以下步骤进行操作：

(1) 创建一个 Web Form 页面，取名 sqldsemployee.aspx。

(2) 从控件工具栏中，选择“数据”选项卡，从中选择 GridView 控件，双击将其加入到页面中。

(3) 从 GridView 控件的任务菜单中选择“新建数据源”选项，如图 10-3 所示。

(4) 这时系统会打开一个数据源配置向导，其第一个配置界面如图 10-4 所示，要求选择数据源类型。从列表中选择“数据库”项，数据源 ID 可以保持默认值不变，单击“确定”按钮进入下一步。

(5) 在如图 10-5 所示的界面中要为数据源指定数据库连接。若前面已经配置过数据库连接，就可以从下拉列表框中选择合适的连接。本例中假定未曾配置过数据库连接，单击“新建连接”按钮，打开如图 10-6 所示的添加数据库连接对话框。

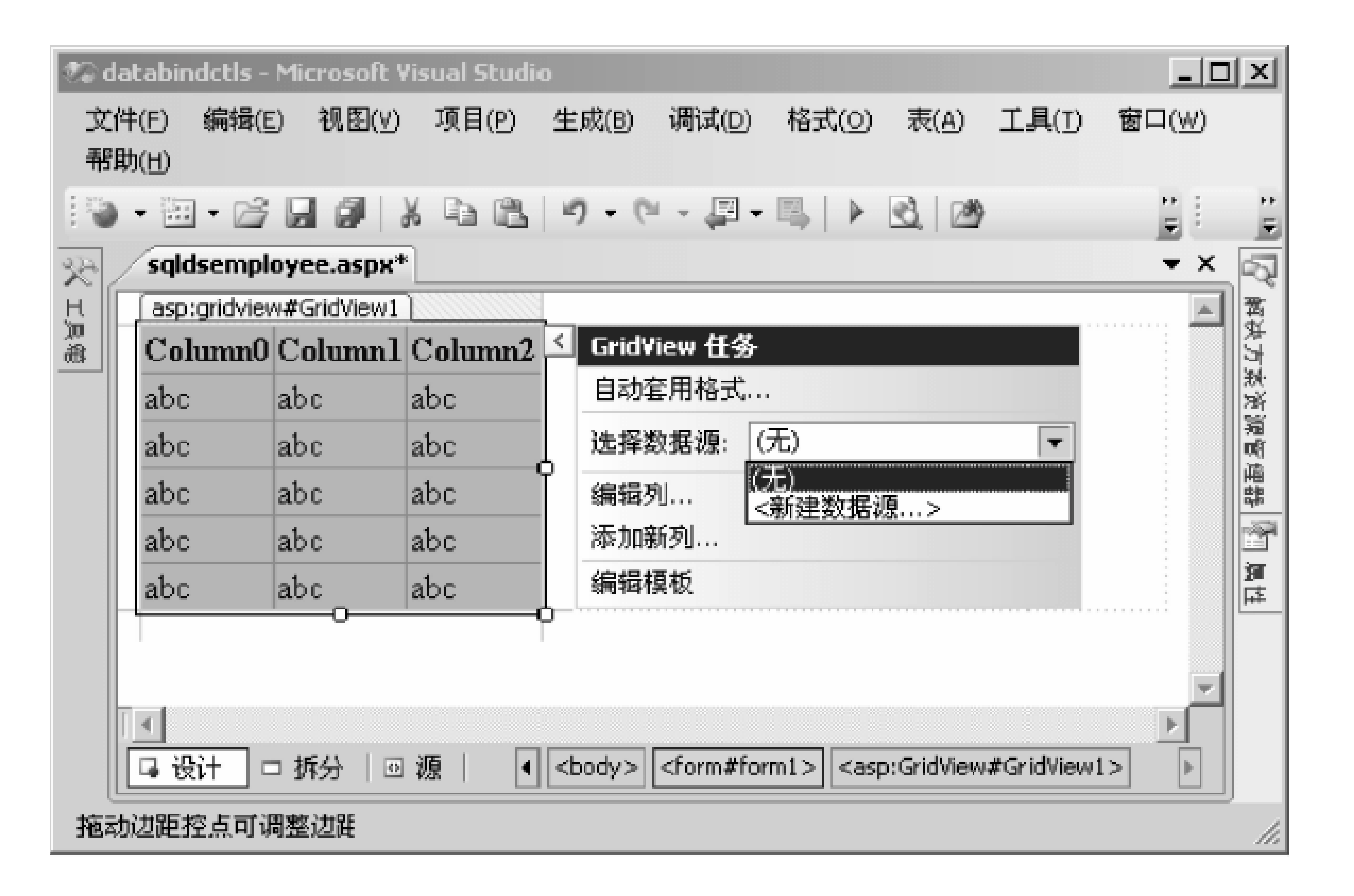


图 10-3 为 GridView 控件新建数据源



图 10-4 数据源配置向导第一步



图 10-5 为数据源指定数据库连接界面

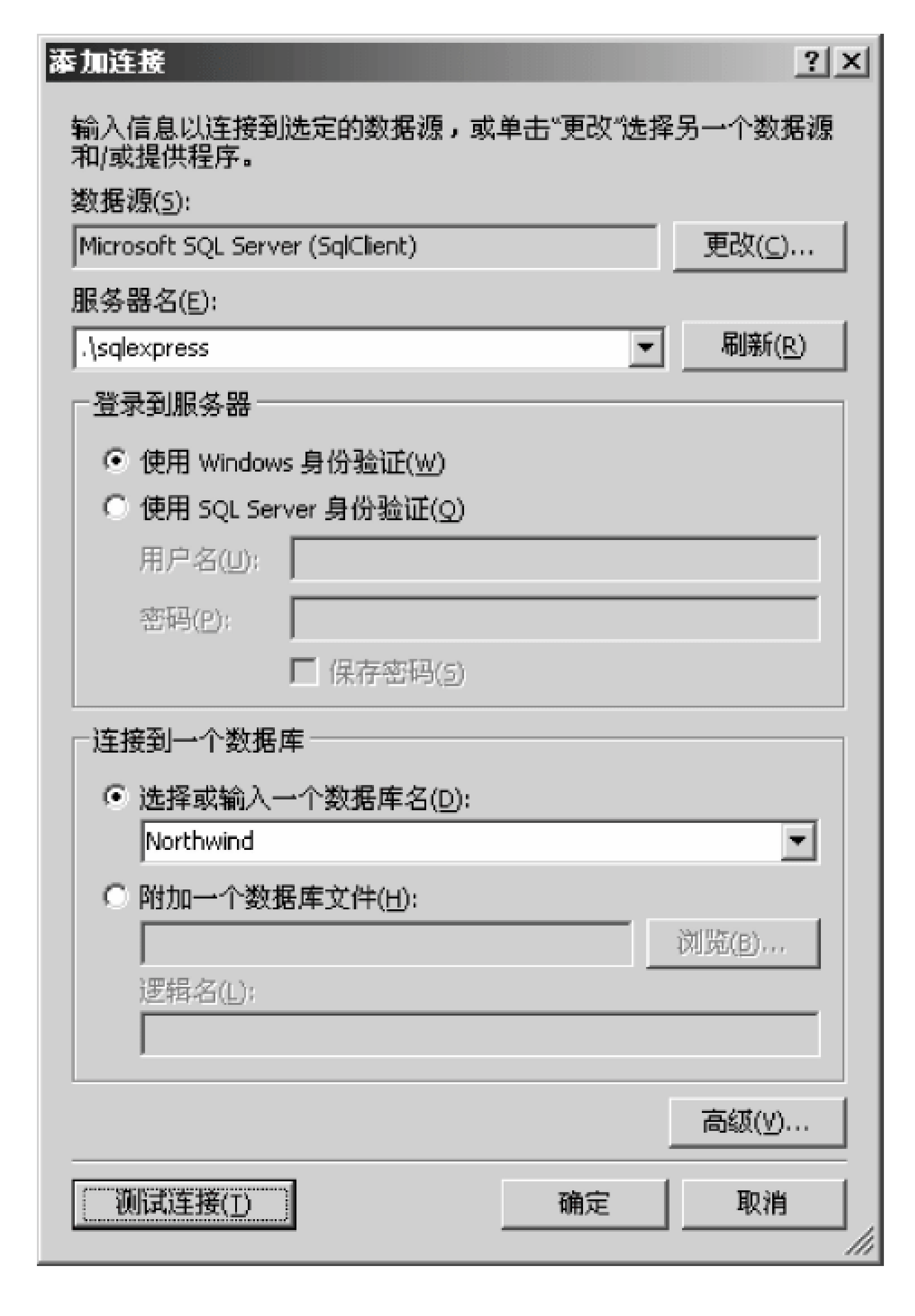


图 10-6 "添加连接"对话框

(6) 在添加连接对话框中，选择数据源的类型为 SqlClient，输入 SQL Server 实例名(本例使用 SQL Server Express 版，默认实例名为 sqlexpress，前面的"．\"代表本机，也可以用 localhost\sqlexpress 等形式)，选择登录方式为"Windows 身份验证"，单击"测试连接"按钮以测试连接的可用性。若提示"测试连接成功"，则说明数据库连接的配置没有问题，从数据库名下拉列表框中选择 NorthWind 数据库，点击"确定"按钮返回选择数据库连接对话框。如图 10-7 所示，这时系统已自动生成了数据库连接，并填入了连接字符串。



图 10-7 选择数据库连接对话框

(7) 单击“下一步”继续，进入如图 10-8 所示的界面，提示将数据库连接字符串保存到配置文件中，勾选“是，将此连接另存为”复选框，单击“下一步”按钮，系统会将连接字符串以指定的键名保存到 Web.config 文件中，并打开如图 10-9 所示的界面。

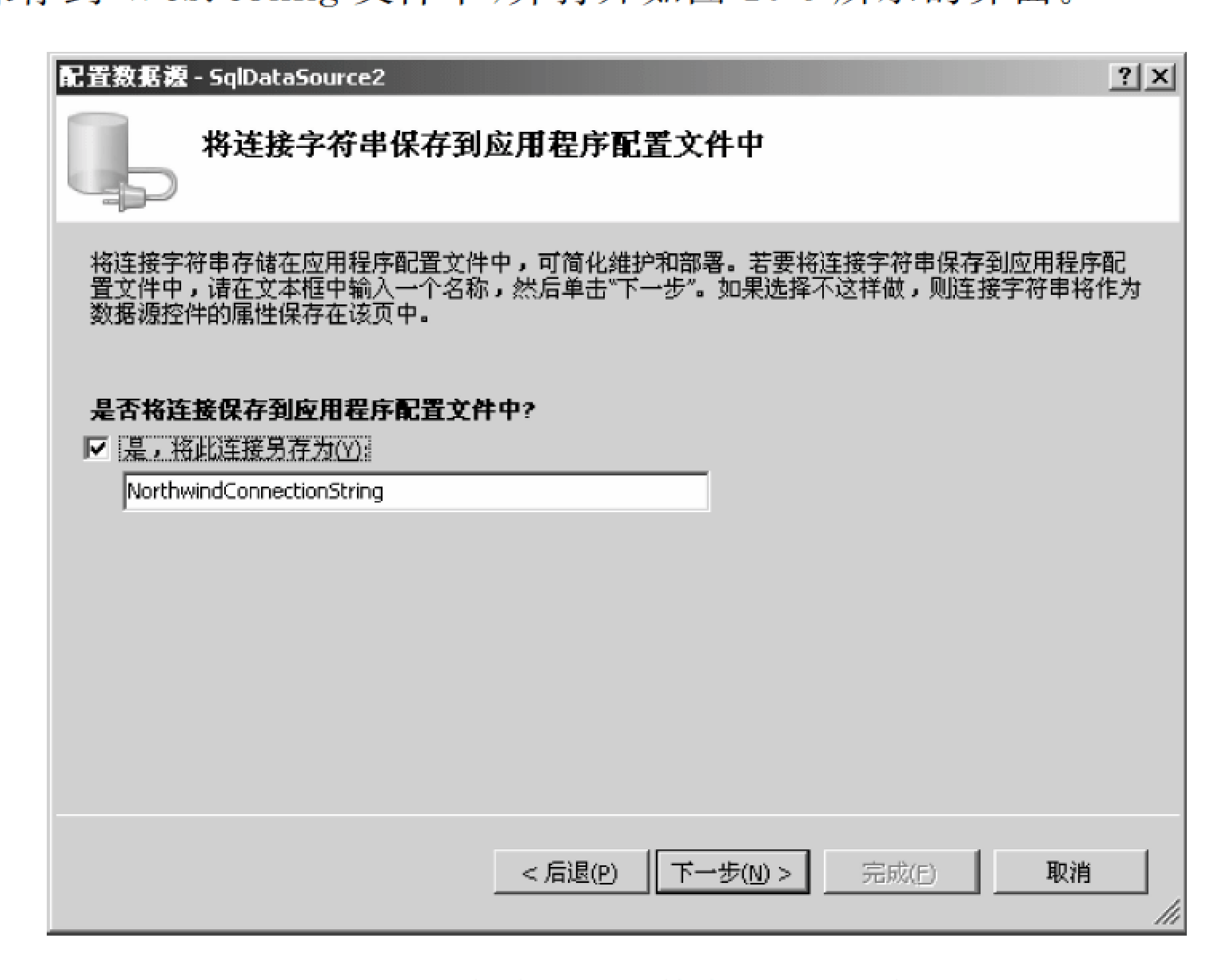


图 10-8　保存连接字符串到配置文件



图 10-9　“配置 Select 语句”对话框

(8) 为数据源配置检索命令，有两种方式：一是直接指定一条 SQL 语句或一个存储过程；二是根据用户的选择由系统自动生成一条 SQL，这里采用后一种方式。先选择“指定来自表或视图的列”单选按钮，并从下面的名称下拉框中选择 Employees 表，这样该表中的所有字段会在下面的列表框中显示出来；在字段名列表框中勾选需要的字段，可以看到，系统

自动生成了 Select 语句并在下面的文本框中显示出来。

(9) 如果需要,可以继续单击 Where 按钮,生成数据过滤条件,或单击 Order By 按钮,生成数据排序子句。

(10) 单击"高级"按钮,可以打开高级 SQL 生成选项对话框,如图 10-10 所示,勾选"生成 INSERT、UPDATE、DELETE 语句"复选框,这样系统将自动为数据源生成三条更新语句。



图 10-10 "高级 SQL 生成选项"对话框

(11) 在图 10-9 中,单击"下一步"按钮,打开测试查询对话框,这里可以测试刚才生成的 SQL 语句的执行情况,单击"测试查询"按钮,可以看到从库中提取的数据显示在网格中。

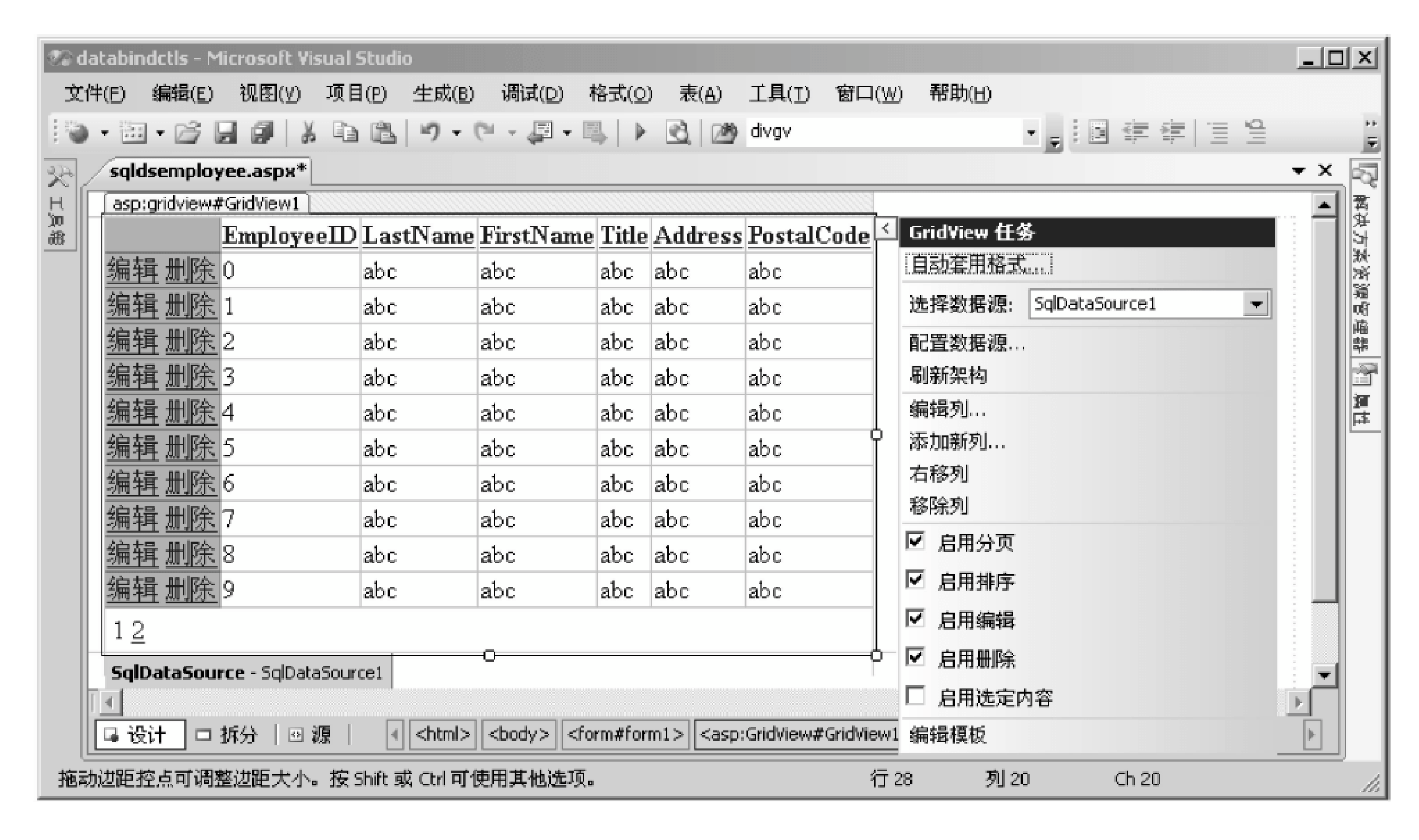


图 10-11 配置 GridView

(12) 若数据没有问题,单击"完成"按钮结束配置向导。可以看到,GridView 已自动添加了从数据源中获取的列。

(13) 再从 GridView 的智能标记菜单中选择"启用分页"、"启用排序"、"启用编辑"、"启用删除"等选项,如图 10-11 所示。可以看到,GridView 的显示样式也随着发生变化。若觉得网格的外观不够好看,还可以从其智能标记菜单中选择"自动套用格式"项来改变其外观。

至此，所有的配置工作已经完成，程序可以运行了。在 VS 的解决方案资源管理器中右击 sqldsemployee.aspx 页面，从弹出的快捷菜单中选择"在浏览器中查看"项，可以看到程序的运行效果，如图 10-12 所示。

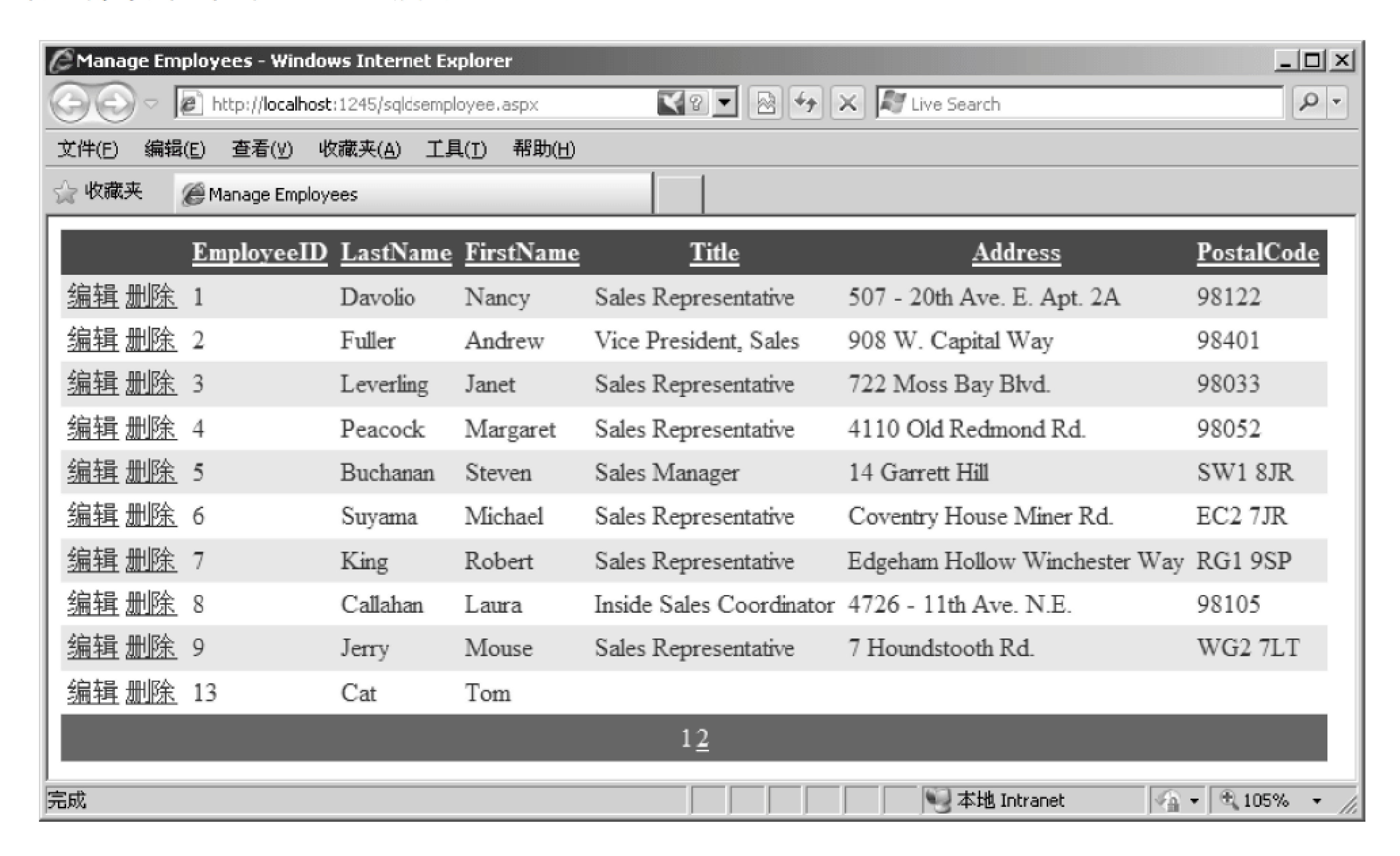


图 10-12　员工管理程序的运行效果

在程序运行界面上单击各列的标题，体验排序的效果；单击页面底部的页码按钮，体验数据分页显示的效果；还可以单击网格左侧的编辑和删除按钮，尝试更新数据，所有的功能都可以正常运行，至此还没有编写一行代码，这就是数据源控件和数据绑定技术带来的巨大便利。

所有的数据源控件都实现了 IDataSource 接口，.NET 框架中主要的数据源控件如表 10-2 所示。

表 10-2　NET 框架中的主要数据源控件

控　件　名	作　　用
SqlDataSource	代表使用 ADO.NET 提供程序连接的关系数据库中得数据，支持使用 SqlClient、OleDb、Odbc 或 OracleClient 连接到的任何关系数据库
ObjectDataSource	代表多层体系结构中的中间层对象。较复杂的应用程序通常将表示层同业务层分开，并在业务对象中封装处理逻辑，ObjectDataSource 使开发人员能够在 n 层体系结构的应用程序中使用数据源控件
AccessDataSource	代表使用 Microsoft Access 数据库的数据源控件，是一种文件数据源
XmlDataSource	是向数据绑定控件提供 XML 数据的数据源控件，可以获取分层数据或表格数据，通常用于显示只读方案中的分层 XML 数据
SiteMapDataSource	是站点地图数据的数据源，可以使 TreeView、Menu 等控件绑定到分层的站点地图数据

10.2.2 使用 SqlDataSource 控件

使用 SqlDataSource 控件可以连接到任何拥有 ADO.NET 数据提供程序的数据源，包括 SQL Server、Oracle 以及基于 OLE DB 或 ODBC 的数据源。

从本质上看，SqlDataSource 会根据配置自动创建 Connection 对象、Command 对象及 DataReader 对象等，以完成各项数据访问操作，这就需要配置一系列参数，包括数据库连接字符串、数据增删改查命令及各种命令参数等。

1. 配置连接字符串

数据库连接字符串可以硬编码到 SqlDataSource 标记中，但推荐的方法是将其保存在配置文件中，然后在 SqlDataSource 中引用。例如，在 web. config 文件中配置如下的连接串：

```
<connectionStrings>
    <add name = "connstr" connectionString = "Data Source = .\sqlexpress;
    Initial Catalog = northwind; Integrated Security = True"
    providerName = "System.Data.SqlClient"/>
</connectionStrings>
```

那么，在 SqlDataSource 标记中，可以按如下的方式引用它：

```
<asp:SqlDataSource ConnectionString = "<% $ ConnectionStrings:connstr %>" … />
```

2. 执行查询命令

SqlDataSource 依靠 4 个命令对象实现数据库的增删改查操作，其命令逻辑由 4 个属性提供，即 SelectCommand、InsertCommand、UpdateCommand 及 DeleteCommand。它们都接收一个命令字符串，该字符串可以是一条 SQL 语句，也可以是一个存储过程的名字，由其 SelectCommandType、InsertCommandType、UpdateCommandType 及 DeleteCommandType 属性值确定，为 StoredProcedure 表示存储过程，为 Text 表示 SQL 语句。

下面是一个完整的 SqlDataSource 定义，可以从 Employees 表读取数据。

```
<asp:SqlDataSource ID = "dsEmployees" runat = "server"
        ConnectionString = "<% $ ConnectionStrings:connstr %>"
        SelectCommand = "SELECT EmployeeID, LastName, FirstName FROM Employees"
</asp:SqlDataSource>
```

既可以在源代码视图中手工建立数据源，也可以在设计视图下利用向导来建立数据源。在控件工具栏的数据标签下选择 SqlDataSource 控件添加到页面上，然后单击该控件，从智能标记中选择"配置数据源"，按向导的提示完成配置，系统即可自动生成数据源的代码。

在各个命令对象中，通常都要使用命令参数以提高命令的灵活性，请看如下示例。

例 10-4 根据居住地查询员工的基本信息。

这里要使用主从表，主表提供员工居住地信息，从表提供居住在某地的员工信息。这样该示例中需要定义两个数据源，一个提供主表数据，另一个提供从表数据。

下面是主表数据源的定义：

```
<asp:SqlDataSource ID = "dsCity" runat = "server"
        ConnectionString = "<% $ ConnectionStrings:connstr %>"
        SelectCommand = "select distinct city from employees">
</asp:SqlDataSource>
```

在页面上添加一个下拉列表框 ddlCity，使用 dsCity 数据源填充它，并将其"自动回发"属性设置为真。代码如下：

```
<asp:DropDownList ID = "ddlCity" runat = "server" AutoPostBack = "True"
         DataSourceID = "dsCity" DataTextField = "city" DataValueField = "city">
</asp:DropDownList>
```

当选择一个城市后，需要查询居住在该城市的所有员工信息，通过从表数据源来完成，下面是它的定义：

```
<asp:SqlDataSource ID = "dsEmployee" runat = "server"
    ConnectionString = "<% $ ConnectionStrings:connstr %>"
    SelectCommand = "SELECT EmployeeID, LastName, FirstName, Title, City
    FROM Employees WHERE (City = @city)">
    <SelectParameters>
    <asp:ControlParameter ControlID = "ddlCity" Name = "city" PropertyName = "SelectedValue" />
    </SelectParameters>
</asp:SqlDataSource>
```

该数据源中使用了命令参数(@city)编写查询。可以定义多个参数，但必须把它们都映射到某个值。本例中@city 参数的值从 ddlcity 控件的 SelectedValue 属性获得，所以使用了 ControlParameter 参数，还可以选择从 QueryString、Session、Cookie、Form 等多种来源中获取参数值。

最后，向页面上添加一个 GridView 控件来显示数据源 dsEmployee 中的数据，代码如下：

```
<asp:GridView ID = "gvEmployee" runat = "server" DataKeyNames = "EmployeeID"
            DataSourceID = "dsEmployee">
</asp:GridView>
```

运行程序，效果如图 10-13 所示。

3. 执行更新命令

SqlDataSource 还支持 Insert、Update、Delete 等数据更新命令的执行，方法是定义 InsertCommand、UpdateCommand 和 DeleteCommand。下面的示例中定义了一个可更新员工信息的数据源。

```
<asp:SqlDataSource ID = "dsEmployee" runat = "server"
      ConnectionString = "<% $ ConnectionStrings:NorthwindConnectionString %>"
      SelectCommand = "SELECT EmployeeID, LastName, FirstName, Title, City FROM Employees"
      UpdateCommand = "UPDATE Employees SET LastName = @LastName, FirstName = @FirstName,
      Title = @Title, City = @City WHERE (EmployeeID = @EmployeeID)">
</asp:SqlDataSource>
```



EmployeeID	LastName	FirstName	Title	City
5	Buchanan	Steven	Sales Manager	London
6	Suyama	Michael	Sales Representative	London
7	King	Robert	Sales Representative	London
9	Jerry	Mouse	Sales Representative	London

图 10-13　根据居住地查询员工信息的网页

这里,更新命令中的参数名字不是随便起的,而是和查询命令中指定的字段名相一致,只在前面加上了@符号,这样就不用再专门定义参数了。

向页面上添加一个 GridView 控件,设置其数据源为 dsEmployee,并在其智能标记中选择“启用编辑”项,这样网格的最左侧将出现一个编辑列,单击某行的编辑按钮后可以对该行数据进行更改,如图 10-14 所示。当单击“更新”按钮时,GridView 会自动将各列的值传递给 DataSource,从而填充 UpdateCommand 的各个参数,将结果保存到数据库。

使用数据源控件执行插入、删除操作的方法与更新操作类似,这里不再详细说明。

使用数据源更新数据 - Windows Internet Explorer
http://localhost:1245/dsUpdate.aspx
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
收藏夹　使用数据源更新数据

	EmployeeID	LastName	FirstName	Title	City
编辑	1	Davolio	Nancy	Sales Representative	Seattle
更新 取消	2	Fuller	Andrew	Vice President, Sales	Tacoma
编辑	3	Leverling	Janet	Sales Representative	Kirkland
编辑	4	Peacock	Margaret	Sales Representative	Redmond
编辑	5	Buchanan	Steven	Sales Manager	London
编辑	6	Suyama	Michael	Sales Representative	London
编辑	7	King	Robert	Sales Representative	London
编辑	8	Callahan	Laura	Inside Sales Coordinator	Seattle
编辑	9	Jerry	Mouse	Sales Representative	London
编辑	13	Cat	Tom		
编辑	14	Mouse	Jerry		

本地 Intranet　105%

图 10-14　使用数据源更新数据库的页面

10.2.3　使用 ObjectDataSource 控件

使用 SqlDataSource 控件通常可以节省大量的数据访问代码,但 SqlDataSource 也不是万能的,它牺牲了很多的效率和灵活性。在大型的 Web 应用中,通常采用 n 层体系结构的模式,页面中不会硬编码 SQL 语句,而是调用业务层或数据访问层对象的方法实现数据处理,这种情况下就要使用 ObjectDataSource 获得更大的灵活性。

1. 使用 ObjectDataSource 执行查询

在与数据打交道时，推荐的做法是将数据访问逻辑从表示层分离出来，形成专门的数据访问层(DAL)，由页面调用数据访问层对象的方法来操作数据库；很多时候还需要在表示层和数据访问层之间再增加一个业务层，负责处理核心的业务逻辑，这样就形成了三层体系结构。在多层体系结构中，各层之间通常使用业务对象(或称值对象，只封装业务数据，不包含处理代码)来传递数据。这种应用场合中，就要使用 ObjectDataSource 来连接前端的表示层与后端的业务层或数据访问层。

例 10-5 使用 ObjectDataSource 控件开发员工信息查询程序。

在多层结构的应用程序中，需要先定义业务对象类和数据访问类。这些类通常放在专门的类库项目中，也可以在网站的应用程序代码目录下定义，本例使用第二种方式。右击网站项目，从弹出的快捷菜单中选择"添加 ASP.NET 文件夹"项，再选择 APP_CODE 项，即可看到网站下名称为 APP_CODE 的文件夹，后面就在该文件夹下建立应用类。

右击 APP_CODE 文件夹，从弹出的快捷菜单中选择"添加新项"，从打开的"模板"对话框中选择"类"，并输入类名 Employee，单击"添加"按钮创建 Employee 类，然后输入如下代码：

```
public class Employee
{
    private int empid;
    public int Empid
    {
        get{return empid;}
        set{empid = value;}
    }
    private string firstname;
    public string Firstname
    {
        get{return firstname;}
        set{firstname = value;}
    }
    private string lastname;
    public string Lastname
    {
        get{return lastname;}
        set{lastname = value;}
    }
    private string city;
    public string City
    {
        get{return city;}
        set{city = value;}
    }
```

可以看出，在该业务对象中，使用属性封装了 Employee 表中的数据。

下一步是设计数据访问层对象，其基本代码框架如下：

```
public class EmployeeDB
{
    // 从配置文件中读取数据库连接字符串
    string connstr = ConfigurationManager.Connection Strings["connstr"].ConnectionString;
    // 获取所有员工居住的城市
    public List<String> getcitys()
    {
        List<String> list = new List<String>();
        using (SqlConnection conn = new SqlConnection(connstr))
        {
            SqlCommand cmd = conn.CreateCommand();
            cmd.CommandText = "select distinct city from employees";
            conn.Open();
            SqlDataReader dr = cmd.ExecuteReader();
            while (dr.Read()){  list.Add(dr.GetString(0));      }
        }
        dr.Close();     cmd.Dispose();
        return list;
    }
    // 获取居住在某城市的所有员工的信息
    public List<Employee> getemployeesbycity(string city)
    {
        List<Employee> list = new List<Employee>();
        using (SqlConnection conn = new SqlConnection(connstr))
        {
            SqlCommand cmd = conn.CreateCommand();
            cmd.CommandText = "SELECT EmployeeID, LastName, FirstName, City
                  FROM Employees WHERE City = @city";
            cmd.Parameters.AddWithValue("@city", city);
            conn.Open();
            SqlDataReader dr = cmd.ExecuteReader();
            while (dr.Read())
            {
                Employee emp = new Employee();
                emp.Empid = dr.GetInt32(0);
                emp.Lastname = dr.GetString(1);
                emp.Firstname = dr.GetString(2);
                emp.City = dr.GetString(3);
                list.Add(emp);
            }
            dr.Close();
            cmd.Dispose();
        }
          return list;
      }
    // 根据员工号获取某员工的基本信息
    public Employee getemployee(int empid){  …  }
    // 向数据库中插入一个员工的信息
    public int insertemployee(Employee emp){  …  }
    // 从数据库中删除一条员工的信息
```

```
        public int deleteemployee(int empid){   …   }
        // 将员工信息的更改保存到数据库中
        public int updateemployee(int employeeID, string firstName, string lastName, string city)
        {   …   }
    }
```

最后再建立表示层页面，取名 objdsQry. aspx。

在页面上放置一个 ObjectDataSource 控件，取名 dsCity。单击 dsCity 的智能标记，选择配置数据源，打开配置向导。选择 EmployeeDB 类作为其业务对象，再选择 getcitys 方法为其 Select 操作所关联的方法，这样当需要数据时，dsCity 控件会自动构造 EmployeeDB 对象，并调用其 getcitys 方法操作数据库，将结果以 List<String>的形式返回。

在页面上添加一个下拉列表框控件，取名 ddlCity，用来显示城市列表。在 ddlCity 的智能标记中单击“配置数据源”，选择 dsCity 为其数据源。这时运行该页面，可以看到城市下列列表框中已经可以填充数据了。

向页面上添加一个 ObjectDataSource 控件，取名 dsEmployee。从其智能标记中选择“配置数据源”，打开配置向导。第一步是选择业务对象，如图 10-15 所示，从下拉框中选择 EmployeeDB 项。单击“下一步”按钮进入如图 10-16 所示的定义数据方法界面，选择 getemployeesbycity 方法为其数据选择方法。单击“下一步”按钮进入如图 10-17 所示的定义参数页面，这里要为 getemployeesbycity 方法中的参数 city 定义数据来源；选择参数源为 Control，ControlID 为 ddlCity，单击“完成”按钮保存配置。

向页面上添加一个 GridView 控件，取名 gvEmployees，从智能标记中为其选择数据源为 dsEmployee。最后再将 ddlCity 控件的 AutoPostBack 属性设置为 True，至此一个简单的多层结构应用程序已开发完成，可以实现按居住城市查询员工信息的业务逻辑。



图 10-15　选择业务对象界面



图 10-16　定义数据方法界面



图 10-17　定义参数界面

2. 使用 ObjectDataSource 执行更新操作

同 SqlDataSource 控件一样，ObjectDataSource 也提供了对可更新绑定的支持。

首先需要为 ObjectDataSource 控件指定 UpdateMethod，以调用业务类中的更新方法，示例代码如下：

```
<asp:ObjectDataSource ID = "dsEmployee" runat = "server" TypeName = "EmployeeDB"
        SelectMethod = "getemployeesbycity" UpdateMethod = "updateemployee" />
```

更重要的是,UpdateMethod 方法要有正确的方法签名。由于更新、插入、删除操作都要自动从链接的数据绑定控件中获取参数的集合,所以这些参数就一定要和数据访问类中相应的方法参数相匹配,包括参数的数量名称和类型等。

例 10-6 为例 10-5 的示例增加数据更新的功能。

(1) 首先为业务类 EmployeeDB 增加更新员工信息的方法:

```
public void updateemployee(int employeeid, string firstname, string lastname, string city)
{
    using (SqlConnection conn = new SqlConnection(connstr))
    {
        SqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = "UPDATE Employees SET LastName = @last, FirstName = @first,
                              City = @city WHERE EmployeeID = @id";
        cmd.Parameters.AddWithValue("@id", employeeid);
        cmd.Parameters.AddWithValue("@last", lastname);
        cmd.Parameters.AddWithValue("@first", firstname);
        cmd.Parameters.AddWithValue("@city", city);
        conn.Open();
        cmd.ExecuteNonQuery();
        cmd.Dispose();
    }
}
```

(2) 为数据源配置数据更新方法:从 dsEmployee 的智能标记中选择"配置数据源"项,在定义数据方法界面中为 Update 方法选择 EmployeeDB 中的 updateemployee 方法,如图 10-18 所示。



图 10-18 为 dsEmployee 配置数据更新方法

查看源代码可以看到，系统不但为 dsEmployee 定义了 UpdateMethod 属性，还根据 updateemployee 方法的原型，为 dsEmployee 定义了 UpdateParameters 参数集合，代码如下：

```
<UpdateParameters>
    <asp:Parameter Name="employeeid" Type="Int32" />
    <asp:Parameter Name="firstname" Type="String" />
    <asp:Parameter Name="lastname" Type="String" />
    <asp:Parameter Name="city" Type="String" />
</UpdateParameters>
```

(3) 为 GridView 控件配置编辑功能：从 gvEmployee 的智能标记中勾选“启用编辑”复选框，可以看到网格控件的最左侧增加了编辑列。

再次运行程序，单击某员工左侧的“编辑”按钮进入编辑模式，更改数据，再单击“更新”按钮，可以看到更改内容已经保存到了数据库。

由于本例中 GridView 的数据来自于 Employee 的集合，所以当提交编辑时，GridView 会为 Employee 类中的每个属性创建一个参数(包括 EmployeeID、FirstName、LastName 和 City)，并将这些参数加入到 ObjectDataSource 的 UpdateParameters 集合。接着，dsEmployee 在 EmployeeDB 类中查找 updateemployee 方法去执行，实现数据更新。

可以看出，UpdateMethod 方法必须具有和值对象中的属性名字完全相同的参数。例如，下面这个方法是匹配的：

```
public void updateemployee(int employeeid, string firstname, string lastname, string city)
```

而下面这个方法不匹配，因为参数的名字不相同：

```
public void updateemployee(int id, string first, string last, string city)
```

下面这个方法也不匹配，因为使用了额外的参数：

```
public void updateemployee(int employeeid, string firstname, string lastname, string city,
string title)
```

方法匹配的算法不区分大小写，并且也不考虑参数的顺序或数据类型，只是寻找具有相同参数个数以及参数名称的方法。只要有这样的方法，更新就会自动提交，无须编写额外的代码。

如果要使用 ObjectDataSource 执行 Insert 或 Delete 操作，就需要配置其 InsertMethod 和 DeleteMethod，并且在业务类 EmployeeDB 中添加对应的方法，详细操作方法请参阅 MSDN 文档，这里不再说明。

10.3　富数据控件

ASP.NET 提供了几个功能强大的富数据控件，可以帮助用户以最小的代码量实现强大的数据展示、编辑等功能，前面章节中使用过的 GridView 控件就是其中之一，类似这样

的控件还有 ListView、DetailsView 和 FormView 等。

说明：富数据控件发展较快，在 ASP.NET 1.X 中提供了 DataGrid、DataList 和 Repeater 三个控件，后来逐渐被 2.0 及以上版本中的 GridView、DetailsView、FormView 以及 ListView 所取代，1.X 中的原始控件继续保留，但开发人员一般不再使用它们。

10.3.1 GridView 控件

GridView 是一个极为灵活的网格控件，在表的行中显示记录，同时还提供了很多易用的特性，包括数据分页、排序、选择以及编辑等，是一个名副其实的全能型控件。使用 GridView，甚至不用编写任何代码，就能实现很多常用的功能，但这样又会损失很多的灵活性和性能，所以通常都要对 GridView 进行定制及编码。

1. 为 GridView 定义列

使用 GridView 最简单的方法是将其 AutoGenerateColumns 属性设置为 True，这样系统会自动从数据源中获取表格的架构信息，并按各字段出现的先后顺序依次在 GridView 中为所有字段创建列。这样做显然缺少必要的灵活性，如无法改变列的显示顺序或隐藏部分列。为解决这些问题，可以将 AutoGenerateColumns 属性设置为 False，并在 GridView 的<Columns>节中自定义列。

表 10-3 列出了 GridView 支持的几种类型的列，列标签出现的顺序决定了列在 GridView 中的显示顺序。

表 10-3 GridView 中使用的列的类型

列	描　述
BoundField	显示数据源中指定字段的文本
CheckBoxField	对于真/假型字段创建一个选项框显示其状态
ImageField	显示二进制字段中的图像数据
ButtonField	为表格中的每个行创建一个按钮，用于捕获事件并编写代码
CommandField	为表格中的每个行提供选择、编辑等常用功能的按钮
HyperLinkField	为表格中的每个行创建一个超链接，并在链接中显示指定内容
TemplateField	允许自定义模板来显示数据或创建控件，为开发提供最大的灵活性

最基本的列类型是 BoundField，绑定到数据对象的某个字段上。

例如：

```
<asp:BoundField DataField = "EmployeeID" HeaderText = "ID" />
```

这将定义一个绑定列，绑定到数据源中的 EmployeeID 数据项上，标题行显示为 ID。

创建 GridView 后，使用智能标记为其设置数据源，然后单击“刷新架构”，系统会自动为数据源中的所有数据项创建绑定列。用户也可以手工修改列对象的属性，来调整列的标题、显示顺序及其他细节。例如，当暂时不想显示某列时，可以将其 Visible 属性设置为假，这既可以在设计时设置，也可以在运行时通过代码设置。

在绑定列的声明中，可以使用表 10-4 所示的常用属性。

表 10-4　**BoundField 中的常用属性**

属　　性	描　　述
DataField	本列中要显示的数据项的字段名或属性名
DataFormatString	格式化字符串，用于控制本列中数据的显示格式
HeaderText	设置本列的标题文本
HeaderImageUrl	设置本列的标题图像
FooterText	设置本列的脚注文本
SortExpress	排序表达式，用于执行基于该列的排序
ReadOnly	当记录处于编辑模式时，该列是否允许修改，为真表示不允许修改
Vissible	该列是否显示在页面上，为假时不显示

通过 D ataFormatString 属性可以设置列中数据的显示格式，这对日期型及数值型数据的显示非常有用。

例如：

```
<asp:BoundField DataField = "UnitPrice" HeaderText = "Price" DataFormatString = "{0:C}" />
```

格式化字符串通常由一个占位符和格式指示器组成，被包含在一组大括号中。本例中 0 代表要格式化的值，C 表示采用货币格式。常用的格式化字符串如表 10-5 所示。

表 10-5　**常用的格式化字符串**

数据类型	格式化串	作　　用	示　　例
数值型	{0: C}	货币格式表示	＄1,234.50，其中货币符号与地区相关
	{0: E}	科学计数法表示	1.23450E+004
	{0: P}	百分比表示	45.6％
	{0: F?}	固定小数位数	对 123.4，采用{0: F3}格式化为 123.400，而采用{0: F0}则格式化为 123
日期型	{0: d}	使用短日期格式	具体格式取决于区域设置中的短日期格式
	{0: D}	使用长日期格式	具体格式取决于区域设置中的长日期格式
	{0: s}	ISO 标准格式	yyyy-MM-ddTHH:mm:ss，例如 2011-07-20T10:00:23
	{0: M}	月日格式	MMMM dd，例如 January 20
	{0: G}	一般格式	依赖于区域设置，如 10/30/2011 10:00:23 AM

这些格式化字符串不只在 GridView 中使用，在其他很多场合也可以使用。

在许多应用场景中，通常使用 GridView 显示概要数据列表，当单击某数据项时，再导航到一个新页面显示详细数据。这可以通过使用 HyperLinkField 列简单实现，请看如下示例。

例 10-7　显示员工信息列表，当单击某员工时，导航到新页面显示详细信息。

该示例需要两个页面：一是员工概要信息浏览页面；二是员工详细信息显示页面。

(1) 新建员工信息浏览页面 hlfieldexam.aspx，并为其配置一个数据源，代码如下：

```
<asp:SqlDataSource ID = "dsEmployee" runat = "server"
      ConnectionString = "<% $ ConnectionStrings:nwconnstr %>"
```

```
        SelectCommand = "SELECT EmployeeID, LastName + ' ' + FirstName As Name, City
                         FROM Employees">
</asp:SqlDataSource>
```

(2) 在员工信息浏览页面上添加一个 GridView 控件，取名 gvEmployee，并在智能标记中为其选择数据源 dsEmployee。这样，系统会自动为其添加三个绑定字段，分别是 EmployeeID、Name 和 City。

(3) 在 gvEmployee 的智能标记中选择“编辑列”，打开字段设置对话框。在“选定的字段”列表框中将 Name 绑定列删掉。然后在“可用字段”列表框中选择 HyperLinkField，单击“添加”按钮，将其添加到选定字段中，并调整其位置到 EmployeeID 字段的下面。

(4) 单击新添加的 HyperLinkField 字段，在右侧的 HyperLinkField 属性框中设置其关键属性，这里主要有三个属性。

- DataTextField：该字段要显示的列，本例中显示用户名，所以设置为“Name”。注意数据源中将 FirstName 和 LastName 拼接在一起命名为 Name。
- DataNavigateUrlFormatString：设置超链接的 URL 格式，本例中单击超链接时要导航到 empDetail.aspx 页面，同时要传递当前员工的 EmployeeID 字段过去，所以 URL 格式为 empDetail.aspx? id={0}。这里使用“? id={0}”来传递参数。
- DataNavigateUrlFields：设置要传递的参数列表，本例中只传递一个参数，即 EmployeeID 列的值，所以该属性设置为 EmployeeID。

设置完成后，在浏览器中访问该页面，运行效果如图 10-19 所示。当鼠标指向某员工的姓名时，在状态栏能看到超链接指向的 URL 字符串，只是单击该链接还无法跳转到指定页面，因为还没有创建员工详细信息显示页面。

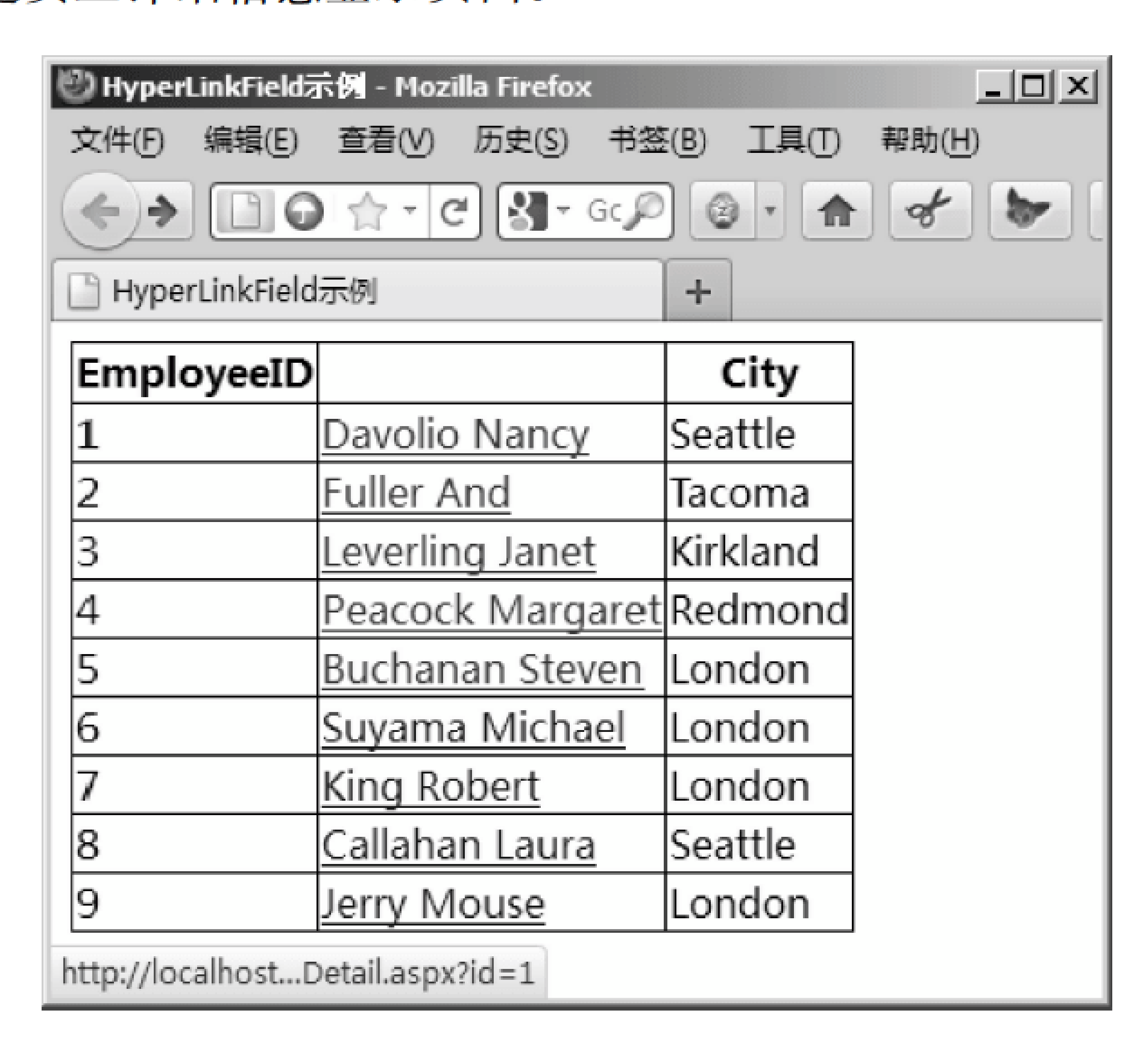


图 10-19　员工概要信息浏览页面的运行效果

(5) 创建员工详细信息显示页面 empDetail.aspx，在页面上添加一个数据源控件，声明如下：

```
<asp:SqlDataSource ID = "dsEmployee" runat = "server"
        ConnectionString = "<% $ ConnectionStrings:nwconnstr %>"
        SelectCommand = "SELECT EmployeeID, LastName, FirstName, Title, BirthDate, Address,
City, Country, HomePhone FROM Employees Where EmployeeID = @ID">
        <SelectParameters>
            <asp:QueryStringParameter Name = "ID" QueryStringField = "id" />
        </SelectParameters>
</asp:SqlDataSource>
```

该控件能够根据 QueryString 中传入的 ID 值从数据库中查询指定员工的详细信息。

(6) 向员工详细信息页面上添加一个 FormView 控件，设置其 DataSourceID 属性为刚才配置的 dsEmployee 数据源，系统会自动根据数据源中的架构为 FormView 创建显示模板。

这时，在员工信息浏览页面单击某员工的姓名，就可以导航到该员工的详细信息页面，如图 10-20 所示。

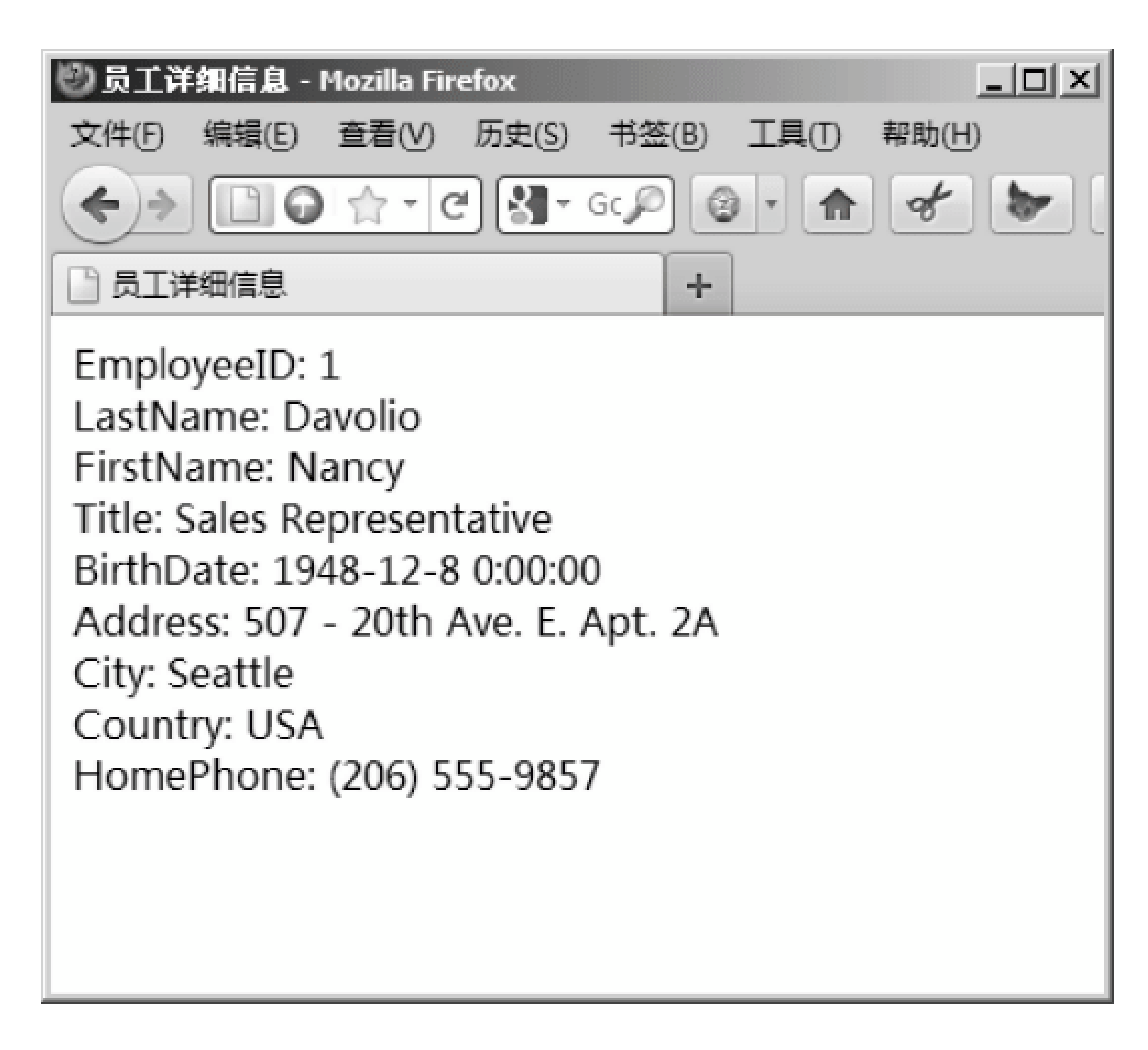


图 10-20 员工的详细信息页面

2. 对 GridView 排序

GridView 控件提供了内置排序功能，无须任何编码。

为启用排序，需要设置 GridView 的 AllowSorting 属性为真，并且为每个可排序的列定义排序表达式。排序表达式一般使用 SQL 查询中 Order By 子句的形式，即包含一个或一系列用逗号分隔的字段名，每个字段名后还可以加上 ASC 或 DESC 以限定升序或降序排列。请看如下示例：

```
<asp:GridView ID = "gvEmployee" runat = "server" AllowSorting = "True"
AutoGenerateColumns = "False" DataKeyNames = "EmployeeID"
DataSourceID = "dsEmployee" >
      <Columns>
        <asp:BoundField DataField = "EmployeeID" HeaderText = "EmployeeID"
              InsertVisible = "False" ReadOnly = "True" SortExpression = "EmployeeID" />
```

```
            <asp:BoundField DataField = "FirstName" HeaderText = "FirstName"
                    SortExpression = "FirstName" />
            <asp:BoundField DataField = "LastName" HeaderText = "LastName"
                    SortExpression = "LastName" />
            <asp:BoundField DataField = "Title" HeaderText = "Title" />
        </Columns>
</asp:GridView>
<asp:SqlDataSource ID = "dsEmployee" runat = "server"
    ConnectionString = "<% $ ConnectionStrings:nwconnstr %>"
    SelectCommand = "Select EmployeeID, FirstName, LastName, Title From Employees">
</asp:SqlDataSource>
```

该页面的运行效果如图10-21所示。由于EmployeeID、FirstName、LastName列都设定了排序表达式，所以这些列的标题栏表现为LinkButton的样式，当单击某标题栏时，表格中的数据会按该列排序显示，若再次单击该列标题，则会按相反顺序重新排列显示。

GridView数据排序 - Windows Internet Explorer

EmployeeID	FirstName	LastName	Title
1	Nancy	Davolio	Sales Representative
2	And	Fuller	Vice President, Sales
3	Janet	Leverling	Sales Representative
4	Margaret	Peacock	Sales Representative
5	Steven	Buchanan	Sales Manager
6	Michael	Suyama	Sales Representative
7	Robert	King	Sales Representative
8	Laura	Callahan	Inside Sales Coordinator
9	Mouseee	Jerry	Sales Representative

图10-21　使用GridView控件排序

说明：　将GridView绑定到数据源控件，并在智能标记中为GridView启用排序后，系统会自动为所有的绑定列启用排序，并自动将排序表达式设置为该列绑定的DataField属性。若想使某列不可排序，只需将该列的SortExpression属性值清空即可。

一般情况下，真正实现排序逻辑的是数据源控件而不是GridView控件。GridView只是展示数据，并提供事件编程的接口；若数据源支持排序，GridView就可以直接利用它，但若数据源不支持排序，用户也可以捕获GridView的Sorting事件并自定义排序方法。

并非所有的数据源控件都支持排序，例如XmlDataSource就不支持，而SqlDataSource和ObjectDataSource就支持。SqlDataSource默认使用DataSet架构(而不是DataReader)保存数据，这样DataSet中的每个DataTable都会链接到一个DataView，通过DataView可对数据进行排序。当用户单击排序列的标题栏时，DataView的Sort属性就被设置为那个列的排序表达式从而实现排序，并将结果绑定到GridView上显示。

3. GridView分页

当GridView中要呈现的记录数量较多时，一般都要启用分页。

GridView 对分页提供内建的支持，可以和数据源控件配合使用实现简单的分页，也可以使用更高效、灵活的方式实现自定义分页。

GridView 提供了几个专为支持分页而设计的属性，如下：

- AllowPaging：是否启用绑定记录的分页，默认为 False。
- PageSize：获取或设置每页中显示的记录数，默认为 10。
- PageIndex：启用分页时，获取或设置当前显示页的序号(从 0 开始编号)。
- PagerSettings：一组分页控件的设置项，决定了分页控件出现的位置及它们包含的文本、图片等。默认这些分页控件显示在页面底部，显示为一系列数字，也可以定制为显示“上页”、“下页”等文字的按钮或图片按钮。
- PageIndexChanging 事件：当单击了分页按钮时发生，可以捕获该事件以定制分页代码。

要使用自动分页，只需将 AllowPaging 属性设置为 True(这时将显示分页控件)，并设置 PageSize 以确定每页显示的行数。

例如：

```
<asp:GridView ID = "gvEmployee" runat = "server" AllowPaging = "True" PageSize = "5" … >
```

自动分页可以和任何实现了 ICollection 接口的数据源一起使用。使用 DataSet 模式的 SqlDataSource 就支持自动分页，而使用 DataReader 模式时则不支持。若自定义的数据访问类能够返回实现了 ICollection 接口的对象，如数组、强类型的集合等，ObjectDataSource 也支持自动分页。

尽管自动分页使用非常方便，但却存在固有缺陷，无法减少数据库查询的数据量。每次当改变页码时，都需要获取所有满足条件的记录，然后绑定指定页的数据，并将其余数据丢弃，这样会造成数据库的沉重负荷以及大量的冗余数据传输。使用数据源控件时，缓存机制会大幅度提高自动分页的性能，减少连接数据库的次数；但若返回成千上万条记录，也会消耗大量的内存。这时就应考虑使用自定义分页。

自定义分页需要编程获取指定页的数据并绑定到 GridView。既可以使用 ADO.NET 数据提供程序访问数据库，也可以定制 ObjectDataSource 访问数据库。使用 ObjectDataSource 时，首先要设置其 EnablePaging 属性为 True，然后还要设置其 StartRowIndexParameterName、MaximumRowsParameterName、SelectCountMethod 属性。

例 10-8 列表显示员工的基本信息，每页显示 5 条记录。

(1) 开发数据访问类。

自定义分页时，必须要有专门的方法计算记录总数，以确定总的页码范围，另外还要有方法能够提取指定的几条记录，这两项任务都要由数据访问类来实现，代码如下：

```
// 计算数据表中的记录总条数
public int countemployees()
{
    int cnt = 0;
    using (SqlConnection conn = new SqlConnection(connstr))
    {
```

```
            SqlCommand cmd = conn.CreateCommand();
            cmd.CommandText = "Select Count(EmployeeID) From Employees ";
            conn.Open();
            cnt = (int)cmd.ExecuteScalar();
            cmd.Dispose();
        }
        return cnt;
}
// 从数据库中查询指定页的记录,start 为起始记录号,count 为每页记录数
public List<Employee> getemployees(int start, int count)
{
    List<Employee> list = new List<Employee>();
    using (SqlConnection conn = new SqlConnection(connstr))
    {
        SqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = "SELECT EmployeeID, LastName, FirstName, Title FROM "
            + "( SELECT EmployeeID, LastName, FirstName, Title, "
            + "ROW_NUMBER() OVER(ORDER BY EmployeeID) as RowNum FROM Employees ) as emps "
            + "Where RowNum Between @start and @end";
        cmd.Parameters.AddWithValue("@start", start);
        cmd.Parameters.AddWithValue("@end", start + count);
        conn.Open();
        SqlDataReader dr = cmd.ExecuteReader();
        while (dr.Read())
        {
            Employee emp = new Employee();
            emp.Employeeid = dr.GetInt32(0);
            emp.Lastname = dr.GetString(1);
            emp.Firstname = dr.GetString(2);
            emp.City = dr.GetString(3);
            list.Add(emp);
        }
        dr.Close();
        cmd.Dispose();
    }
    return list;
}
```

检索数据的方法中使用了比较复杂的 SQL 语句,在子查询中使用 ROW_NUMBER 函数为每一条满足条件的记录生成了行号,然后在主查询中根据行号检索连续的几条记录。

说明:ROW_NUMBER 函数为 SQL SERVER 2005 及以上版本中新增的函数,在低版本的 SQL SERVER 中无法使用。

(2) 声明 ObjectDataSource 数据源,代码如下:

```
<asp:ObjectDataSource ID="dsEmployee" runat="server" TypeName="EmployeeDB"
      EnablePaging="True" SelectCountMethod="countemployees" SelectMethod="getemployees"
      MaximumRowsParameterName="count" StartRowIndexParameterName="start">
</asp:ObjectDataSource>
```

这里，使用 SelectCountMethod 属性指定了计算记录总数的方法，通过 SelectMethod 属性指定了检索数据的方法，最后两个属性为检索方法提供起始记录号和该页的记录数。

(3) 创建 GridView 对象，设置其数据源为 idsEmployee，并启用分页，代码如下：

```
<asp:GridView ID = "gvEmployee" runat = "server" AllowPaging = "True" PageSize = "5"
      DataSourceID = "dsEmployee"  … >
</asp:GridView>
```

运行程序，看到如图 10-22 的页面效果。

Employeeid	Firstname	Lastname	City
1	Nancy	Davolio	Sales Representative
2	And	Fuller	Vice President, Sales
3	Janet	Leverling	Sales Representative
4	Margaret	Peacock	Sales Representative
5	Steven	Buchanan	Sales Manager
1 2			

图 10-22　使用 GridView 控件建立分页

4. 在 GridView 中处理行数据

用户经常要在 GridView 中选择某一行，然后对该行的数据进行处理。要实现这样的功能，需要做两方面的工作：一是在 GridView 中创建命令列或按钮列以触发行事件；二是捕获特定的事件，在事件过程中编写代码进行数据处理。

ButtonField 是一种通用的按钮列，可在 GridView 中创建一列普通的按钮对象，所有按钮都共用同一个命令名(由其 CommandName 属性指定)，用以指定当该按钮被单击时将触发的命令。针对 GridView 的行，通常可以触发以下的命令：

- Delete：删除一行数据。
- Edit：编辑一行数据。
- Update：使用更改的数据执行更新。
- Cancel：放弃更改的数据。
- Select：选择一行数据。

当单击 ButtonField 中的某个按钮时，将触发 RowCommand 事件，开发人员可以在该事件中编写代码，获取命令名，并根据命令执行相应的操作。当命令名为以上 5 条命令之一时，将引发相应的内置事件，包括 RowDeleting、RowDeleted、RowEditing、RowUpdating、RowUpdated、RowCancelingEdit、SelectedIndexChanging、SelectedIndexChanged 等，要处理该命令，就可以选择更合适的事件过程来编写代码，而不必选择通用的 RowCommand 事件过程。

为简化编程，GridView 还提供了 CommandField 列，包装了上面列出的 5 条常用命令。

在创建 CommandField 列时，可以从三组常用命令中选择一种，即编辑、更新、取消命令，或选择命令，或删除命令。

对于 ButtonField 和 CommandField，其按钮都有三种样式供选择，分别是 Button 样式（传统的按钮形状）、Link 样式（超链接的形式）和 Image 样式（图片按钮），可以通过 ButtonType 属性进行设置。

以下通过几个示例说明如何在 GridView 中处理行数据。

例 10-9　显示员工列表，当选择某员工时，在列表下方显示其所有的订单列表。

本例需要创建两个 GridView，一个显示员工列表；另一个显示订单列表。在第一个网格上捕获行选择事件，然后根据所选行的 EmployeeID，在 Order 表中查询订单信息并绑定到第二个网格上显示。

（1）创建 SelectCMD.aspx 页面，添加一个数据源控件以检索员工信息，声明代码如下：

```
<asp:SqlDataSource ID = "dsEmployee" runat = "server"
ConnectionString = "<% $ ConnectionStrings:nwconnstr %>"
SelectCommand = "SELECT EmployeeID, LastName, FirstName, Title FROM Employees" >
</asp:SqlDataSource>
```

（2）创建第一个网格控件，命名为 gvEmployee，设置其数据源为 dsEmployee，系统自动为其创建几个绑定列。再创建第二个网格控件，命名为 gvOrder。

（3）从 gvEmployee 的智能标记中选择“编辑列”，打开如图 10-23 所示的对话框。在可用字段列表框中单击 CommandField 下的“选择”项，单击“添加”按钮，将创建一个 CommandName 属性为 Select 的命令列。移动该列到合适的位置，设置其 HeaderText 属性为“查询订单”。



图 10-23　使用命令列

这时浏览该页面，可以看到 GridView 最左侧的命令列，单击“选择”按钮时没有反应，这是因为还没有处理它的行选择事件。

（4）选择 gvEmployee 控件，在属性窗口中单击“事件”图标，从其内置事件列表中选择

gvEmployee_SelectedIndexChanged 事件(该事件将在 GridView 中选择某行数据时被触发),双击,建立该事件过程,如图 10-24 所示。

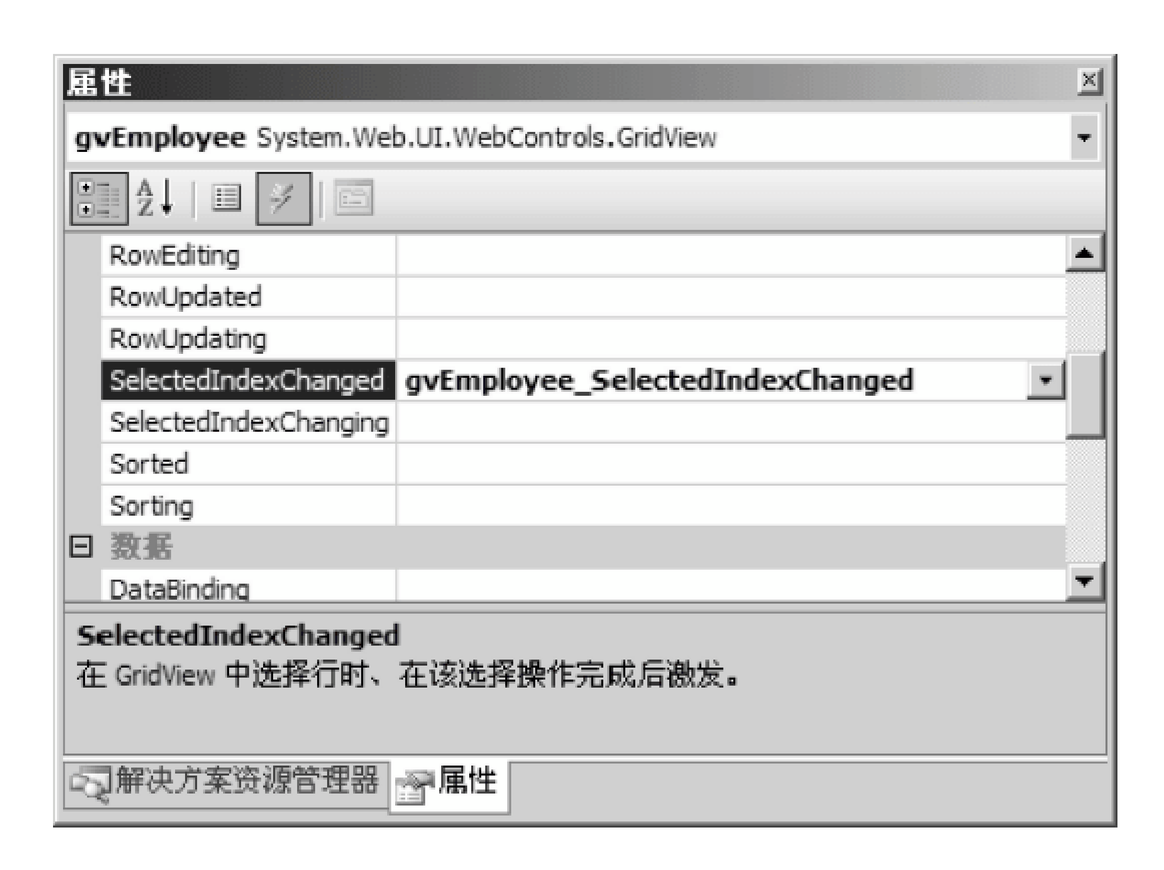


图 10-24　捕获 SelectedIndexChanged 事件

(5) 在打开的代码窗口中,建立如下的事件过程代码:

```
protected void gvEmployee_SelectedIndexChanged(object sender, EventArgs e)
{
      int id = (int)gvEmployee.SelectedDataKey.Value;
      using (SqlConnection conn = new SqlConnection(connstr))
      {
          SqlCommand cmd = conn.CreateCommand();
          cmd.CommandText = "SELECT OrderID, CustomerID, OrderDate
                             FROM Orders Where EmployeeID = @EID";
          cmd.Parameters.AddWithValue("@EID", id);
          conn.Open();
          SqlDataReader dr = cmd.ExecuteReader();
          gvOrder.DataSource = dr
          gvOrder.DataBind();
      }
}
```

再次运行程序,单击某员工前面的"选择"按钮,该员工的订单能够列表显示,如图 10-25 所示。

在 SelectedIndexChanged 事件中,最关键的一步是获取所选员工的 ID 号,可通过两种方式实现。

方式 1:若在 gvEmployee 中设置 EmployeeID 列为主键列,则可以在当前行的主键中取得 EmployeeID,代码如下:

```
int id = (int)gvEmployee.SelectedDataKey.Value;
```

或

```
int id = (int)gvEmployee.SelectedDataKey.Values[0];
```



查询订单	EmployeeID	LastName	FirstName	Title
选择	1	Davolio	Nancy	Sales Representative
选择	2	Fuller	And	Vice President, Sales
选择	3	Leverling	Janet	Sales Representative
选择	4	Peacock	Margaret	Sales Representative
选择	5	Buchanan	Steven	Sales Manager
选择	6	Suyama	Michael	Sales Representative
选择	7	King	Robert	Sales Representative
选择	8	Callahan	Laura	Inside Sales Coordinator
选择	9	Jerry	Mouse	Sales Representative

OrderID	CustomerID	OrderDate
10258	ERNSH	1996-7-17 0:00:00
10270	WARTH	1996-8-1 0:00:00
10275	MAGAA	1996-8-7 0:00:00
10285	QUICK	1996-8-20 0:00:00
10292	TRADH	1996-8-28 0:00:00

图 10-25 显示员工及其订单信息的页面

由于关系表中通常可以有几个字段联合做主键，所以当前选择行的主键应该是多个字段值的集合，上面两种方式都是从这个值集合中取出索引 0 位置的值返回。

方式 2：对于非主键列，可以从当前选择行的指定单元格中获取值，代码如下：

```
string eid = gvEmployee.SelectedRow.Cells[1].Text;
```

使用 SelectedRow 属性可以获得网格中的当前选定行，该行由多个单元格组成，使用 Cells 属性访问单元格集合，指定要访问的单元格序号（从 0 开始编号）即可定位到指定单元格，最后通过 Text 属性得到该单元格的数据。

例 10-10 手工编写代码，实现员工信息的编辑操作。

前面的示例中多使用数据源控件，可以不用编写代码就实现很多功能。但为取得更好的性能和灵活性，往往会使用 ADO.NET 数据提供程序定制代码。

(1) 创建 EditEmployee.aspx 页面，并添加 GridView 控件，声明代码如下：

```
<asp:GridView ID = "gvEmployee" runat = "server" AutoGenerateColumns = "False"
    onrowediting = "gvEmployee_RowEditing"
    onrowupdating = "gvEmployee_RowUpdating">
    onrowcancelingedit = "gvEmployee_RowCancelingEdit"
    <Columns>
        <asp:CommandField HeaderText = "Edit" ShowEditButton = "True" />
        <asp:BoundField DataField = "EmployeeID" HeaderText = "EmployeeID"
            ReadOnly = "True" />
        <asp:BoundField DataField = "FirstName" HeaderText = "FirstName" />
```

```
            <asp:BoundField DataField = "LastName" HeaderText = "LastName" />
            <asp:BoundField DataField = "Title" HeaderText = "Title" />
        </Columns>
    </asp:GridView>
```

注意：这里声明的三个事件过程如下。

Onrowediting：单击 Edit 按钮后触发，通常在这里编码使 GridView 进入编辑模式

Onrowupdating：当编辑完数据并单击 Update 按钮时发生，通常在这里编码将修改保存到数据库中。

Onrowcancelingedit：当编辑过程中单击 Cancel 按钮时发生，通常在这里编写代码，放弃所做的编辑，返回到浏览模式。

(2) 显示所有记录。

为使页面运行时能自动显示所有员工信息码，通常使用 Page_Load 事件，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        bindgv();
    }
}
private void bindgv()
{
    using (SqlConnection conn = new SqlConnection(connstr))
    {
        conn.Open();
        SqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = "SELECT EmployeeID, FirstName, LastName, Title
                           FROM Employees";
        SqlDataReader dr = cmd.ExecuteReader();
        gvEmployee.DataSource = dr;
        gvEmployee.DataBind();
        dr.Close();
        cmd.Dispose();
    }
}
```

注意：在 Page_Load 方法中，要先判断页面是否回发，若不是回发，则检索并绑定员工信息到 GridView。该页面运行效果如图 10-26 所示。

(3) 处理 Edit 按钮事件。

GridView 使用 EditIndex 属性指示哪条记录当前应处于编辑模式，通常情况下该值设置为－1，表示所有记录都处于浏览模式。若要进入编辑模式，只要设置 EditIndex 属性为要编辑记录的索引号，并重新绑定一次数据。核心代码如下：

```
protected void gvEmployee_RowEditing(object sender, GridViewEditEventArgs e)
{
    gvEmployee.EditIndex = e.NewEditIndex;
    bindgv();
}
```

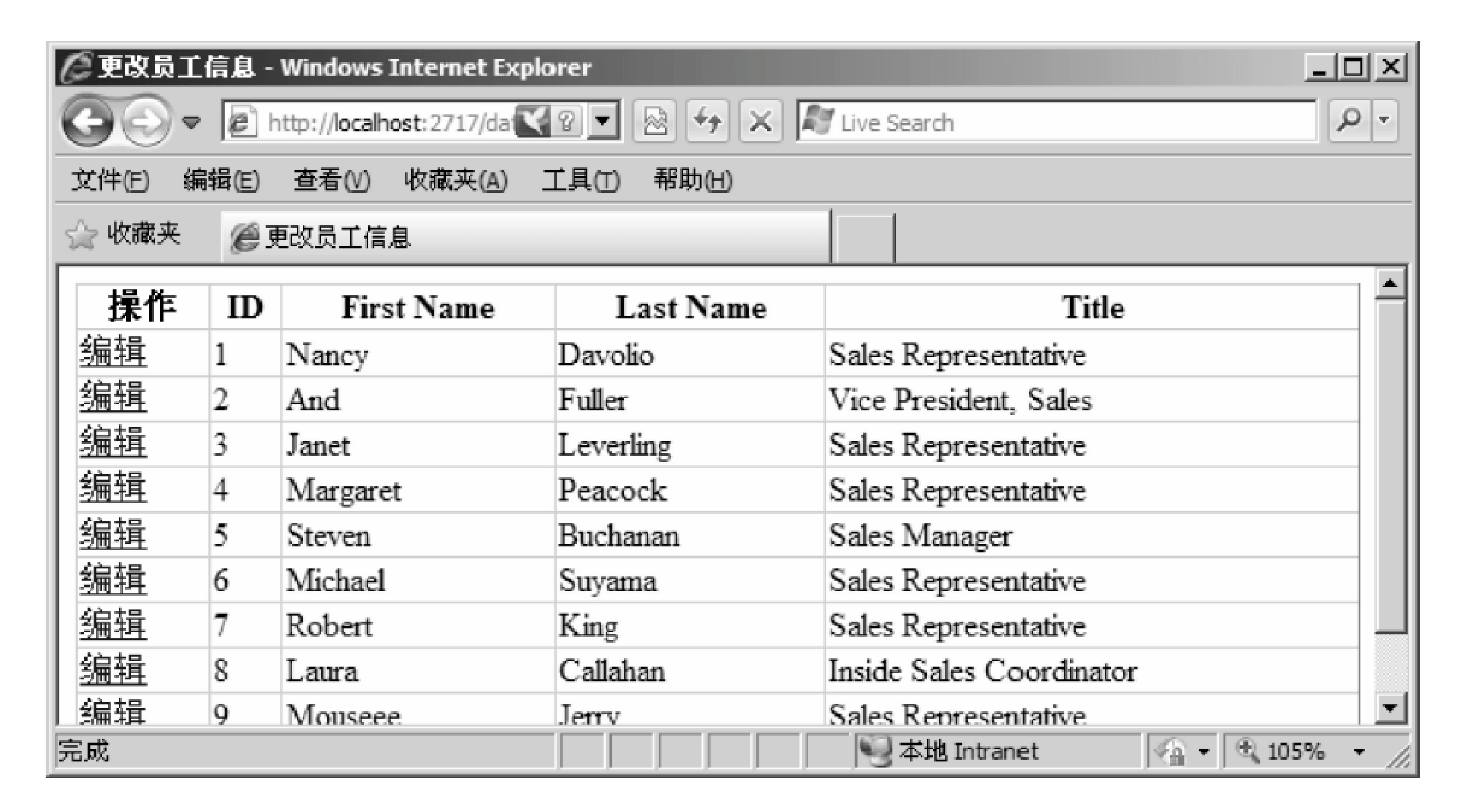


图 10-26　使用 GridView 显示所有员工信息

编辑模式的运行效果如图 10-27 所示。

更改员工信息 - Windows Internet Explorer

http://localhost:2717/da　Live Search

文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)

收藏夹　更改员工信息

操作	ID	First Name	Last Name	Title
更新 取消	1	Nancy	Davolio	Sales Representative
编辑	2	And	Fuller	Vice President, Sales
编辑	3	Janet	Leverling	Sales Representative
编辑	4	Margaret	Peacock	Sales Representative
编辑	5	Steven	Buchanan	Sales Manager
编辑	6	Michael	Suyama	Sales Representative
编辑	7	Robert	King	Sales Representative
编辑	8	Laura	Callahan	Inside Sales Coordinator

完成　本地 Intranet　105%

图 10-27　编辑模式的运行效果

(4) 处理 Cancel 按钮事件。

当用户在编辑模式下单击 Cancel 按钮时，应放弃当前所做修改，并返回到浏览模式，代码如下：

```
protected void gvEmployee_RowCancelingEdit( object sender,
               GridViewCancelEditEventArgs e)
{
          gvEmployee.EditIndex =-1;
          bindgv();
}
```

(5) 处理 Update 按钮事件。

当用户对某条记录编辑完成,单击 Update 按钮时,一般要将更改保存到数据库中,并使 GridView 重新进入浏览模式。核心代码如下:

```
protected void gvEmployee_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    GridViewRow row = gvEmployee.Rows[e.RowIndex];
    string id = row.Cells[1].Text;
    string fname = ((TextBox)row.Cells[2].Controls[0]).Text;
    string lname = ((TextBox)row.Cells[3].Controls[0]).Text;
    string title = ((TextBox)row.Cells[4].Controls[0]).Text;
    updategv(id, fname, lname, title);
}
```

该方法的重点是获取输入的数据。该事件过程中系统自动传入了参数 e,通过它可以获得当前正在编辑的记录的索引号,进而可以在网格中找到该行。GridView 中的每一行都是由多个单元格构成,所以要在单元格中检索数据。

对于后三个字段,由于要进行编辑,数据绑定时系统会在各自单元格中分别创建文本框(或复选框)来绑定数据;这样,就可以从单元格中还原文本框,进而得到其文本;注意,还原文本框时要使用强制类型转换。由于 EmployeeID 为主键,其值不允许修改,系统不会为其创建文本框对象,可以通过单元格的 Text 属性直接获取其值,或通过主键集合获取该行的主键值,这里使用了第一种方法。

然后保存数据,并进行页面更新,代码如下:

```
private void updategv(string id, string fname, string lname, string title)
{
    using (SqlConnection conn = new SqlConnection(connstr))
    {
        conn.Open();
        SqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = "Update Employees "
                set FirstName=@fname, LastName=@lname, Title=@title
                Where EmployeeID=@id";
        cmd.Parameters.AddWithValue("@id", id);
        cmd.Parameters.AddWithValue("@fname",fname);
        cmd.Parameters.AddWithValue("@lname", lname);
        cmd.Parameters.AddWithValue("@title", title);
        cmd.ExecuteNonQuery();
        cmd.CommandText = @"SELECT EmployeeID, FirstName, LastName, Title
                            FROM Employees";
        cmd.Parameters.Clear();
        SqlDataReader dr = cmd.ExecuteReader();
```

```
            gvEmployee.DataSource = dr;

            gvEmployee.EditIndex =-1;
            gvEmployee.DataBind();
            dr.Close();
            cmd.Dispose();
        }
}
```

注意：数据保存后一定要重新绑定一次 GridView，否则将无法显示任何数据。

5. 在 GridView 中使用模板列

使用数据绑定控件时，由于要显示的数据通常包含多条结构类似的记录，因此，经常使用模板(Template)来指定单条记录的显示格式，然后数据绑定控件自动将这一定义好的模板应用于所有要显示的记录。

GridView 控件大量使用模板，前面的示例中使用了多种系统内置的模板。若要完全按自己的想法来布置页面，就要使用模板列(Template Field)。请看如下的示例。

例 10-11 列表显示所有员工的详细信息。

传统的 GridView 模板都是在一行中显示一条记录，但由于员工信息数据项较多，而且有的数据项很长，难以在表行中显示出来，这时就要定制模板，声明代码如下：

```
<asp:GridView ID="gvEmployee" runat="server" AutoGenerateColumns="False"
    GridLines="None">
    <HeaderStyle BackColor="Silver" />
    <Columns>
        <asp:TemplateField HeaderText="Employees">
        <ItemTemplate>
            <b>
            <%# Eval("EmployeeID") %>-
            <%# Eval("FirstName") %>
            <%# Eval("LastName") %>
            <hr /></b>
            <small>
            <%# Eval("Address") %><br />
            <%# Eval("City") %>, <%# Eval("Country") %><br />
            <%# Eval("HomePhone") %><br />
            <%# Eval("Notes") %><br /><br />
            </small>
        </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

图 10-28 显示了该模板的运行效果。

在模板列中可以加入任意的 HTML 元素以及数据绑定表达式等，完全可以按照自己的方式布置一切。当执行数据绑定时，GridView 会从数据源中获取数据并循环遍历这些数据项目的集合，它为每个项目处理 ItemTemplate，计算其中的数据绑定表达式并将值插入

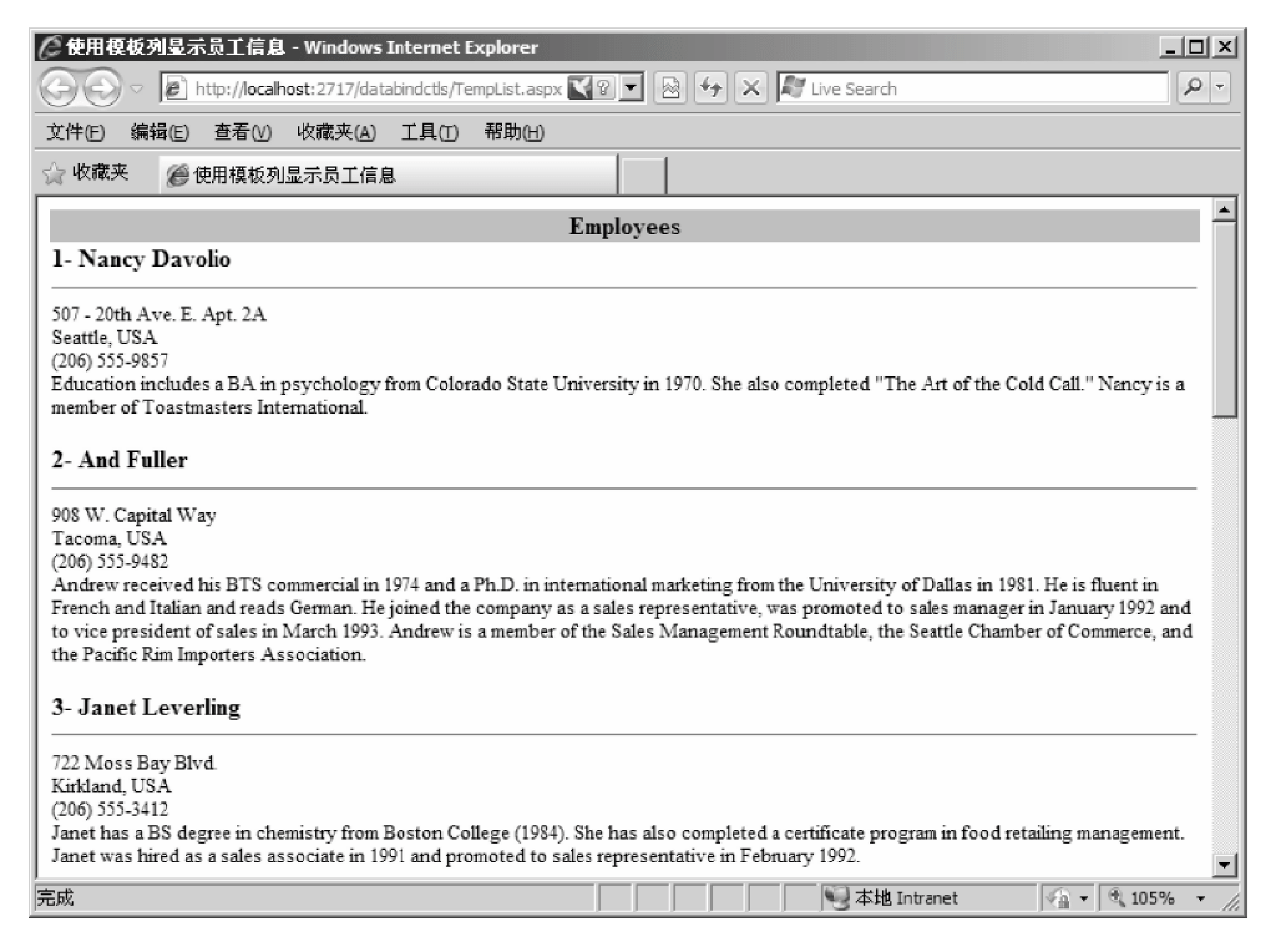


图 10-28　列表显示所有员工的详细信息

到 HTML 中。

在本例中，使用了 Eval 方法计算数据绑定表达式的值，这是 System.Web.UI.DataBinder 类的一个静态方法，为开发带来了极大的便利。它自动读取绑定到当前行的数据项，使用反射机制找到匹配的字段或属性并获取值。另外，Eval 方法中还允许接收格式化字符串以控制数据的显示格式。

例如：

```
<%# Eval("BirthDate", "{0 : MM/dd/yy}") %>
```

可以针对不同的场景定义不同的模板。例如，针对浏览定义一个只读的模板，针对浏览中的交替项显示不同的样式，为编辑状态制定可读写的模板等。多数数据绑定控件都提供了相应的方法，能够在不同状态之间切换，并自动加载相应的模板。

常用的模板类型如下：

- ItemTemplate：普通项目模板，用于浏览状态。
- AlternatingItemTemplate：交替项模板，浏览状态下可使用该模板使相邻的两行数据采用不同的样式显示，增强对比度。
- EditItemTemplate：编辑项模板，只对当前 EditIndex 指向的记录起作用。
- HeaderTemplate：表头的模板。
- FooterTemplate：页脚的模板。

下面的示例演示了使用 EditItemTemplate 进行数据编辑的功能。

例 10-12 使用模板列编辑员工的称谓及备注信息。

此例中，称谓只能从“Mr.”、“Dr.”、“Ms.”及“Mrs.”中选择一个，备注信息较长，要使用多行文本框编辑。

本例若对称谓和备注字段使用绑定列，那么编辑模式下将显示为单行文本框，无法达到题目要求，所以这里使用模板列，分别定制 ItemTemplate 和 EditItemTemplate 模板。

页面声明代码如下：

```
<asp:GridView ID="gvemployee" runat="server" AutoGenerateColumns="False"
    DataSourceID="dsEmployee" Width="100%" GridLines="None">
    <Columns>
        <asp:TemplateField HeaderText="Edit Employees Information">
            <ItemTemplate><b>
                <%# Eval("EmployeeID") %> - <%# Eval("TitleOfCourtesy") %>
                <%# Eval("FirstName") %>
                <%# Eval("LastName") %><hr /></b>
                <%# Eval("Address") %><br />
                <%# Eval("HomePhone") %><br />
                <%# Eval("Notes") %><br /><br />
            </ItemTemplate>
            <EditItemTemplate><b>
                <asp:Label ID="lblid" runat="server"
                    Text='<%# Bind("EmployeeID") %>' /> -
                <asp:DropDownList ID="ddltitle" runat="server"
                    DataSource="<%# CourtesyTitle %>"
                    SelectedValue='<%# Bind("TitleOfCourtesy") %>'>
                </asp:DropDownList>
                <%# Eval("FirstName") %>
                <%# Eval("LastName") %><hr /></b>
                <%# Eval("Address") %><br />
                <%# Eval("HomePhone") %><br />
                <asp:TextBox ID="tbxNotes" runat="server"
                    Text='<%# Bind("Notes") %>'
                    TextMode="MultiLine"></asp:TextBox><br />
            </EditItemTemplate>
        </asp:TemplateField>
        <asp:CommandField HeaderText="操作" ShowEditButton="True">
            <HeaderStyle Width="10%" /><ItemStyle HorizontalAlign="Center" />
        </asp:CommandField>
    </Columns>
</asp:GridView>
<asp:SqlDataSource ID="dsEmployee" runat="server"
    ConnectionString="<%$ ConnectionStrings:nwconnstr %>"
    SelectCommand="SELECT EmployeeID, FirstName, LastName, TitleOfCourtesy,
                   Address,HomePhone,Notes FROM Employees"
    UpdateCommand="Update Employees set TitleOfCourtesy=@titleofcourtesy,
                   Notes=@notes Where EmployeeID=@employeeid">
</asp:SqlDataSource>
```

可以看到，在 EditItemTemplate 中，三个关键字段的绑定方式和 ItemTemplate 中有很

大的区别。ItemTemplate 中的数据绑定是单向的，只需将数据源的数据绑定到控件上显示，不需回传控件的值给数据源，这种绑定使用 Eval 方法即可。在 EditItemTemplate 中，有些控件还要将更改后的值回传给 DataSource 控件，以便能够以这些数据为参数执行更新语句，这种双向传值的情境下必须使用 Bind 方法绑定数据。当 GridView 提交更新时，只提交 Bind 方法绑定的参数，所以 Update 语句的所有参数必须以 Bind 方法绑定，否则将无法接收到值。

对于 EmployeeID 字段，由于只是显示，不允许更改，所以只需定义一个标签控件，并将字段值绑定到标签的 Text 属性上。

对于称谓字段，要求只能选择而不能输入，所以要在模板中为其创建下拉框。下拉框中的选项又如何来呢？本例中通过 DataSource 属性，将一个字符串数组绑定，而该字符串数组可以通过页面的属性或方法定义，本例中使用了属性，代码如下：

```
protected string[] TitleOfCourtesy
{
    get
    {
        return new string[] { "Mr.", "Dr.", "Ms.", "Mrs."};
    }
}
```

下拉框的数据绑定后，还要设置员工的当前称谓信息，将当前员工的称谓字段值绑定到下拉框的 SelectedValue 属性上即可。

对于备注字段，创建了多行文本框，并将字段值绑定到文本框的 Text 属性上。

为支持数据更新，在 GridView 中还定义了 CommandField 列。页面运行效果如图 10-29 所示。

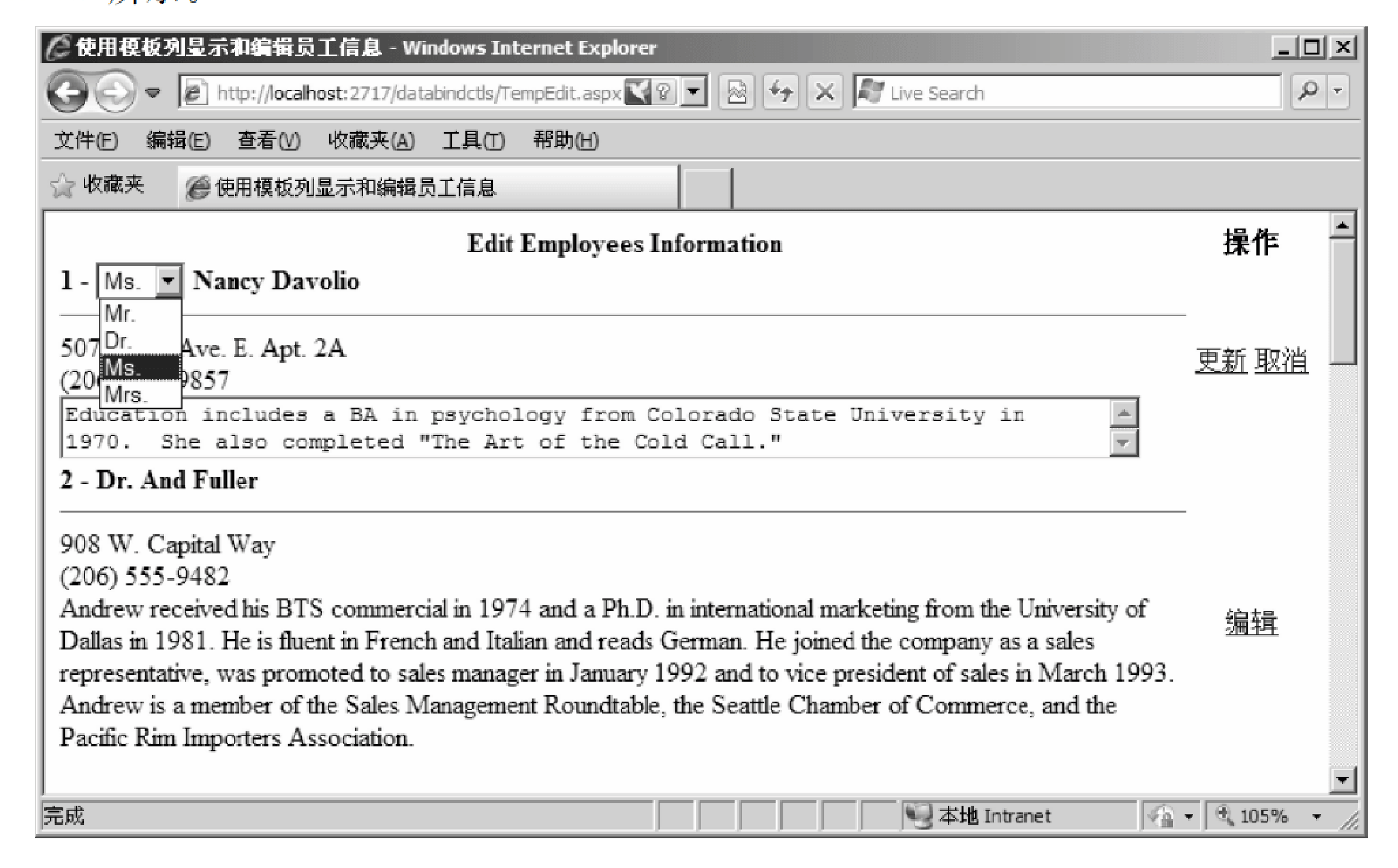


图 10-29 使用模板列编辑信息

若不想使用数据源控件，而要在 GridView 的 RowUpdating 事件过程中编码实现数据更新，那么，可以在 GridView 行上调用 FindControl 方法查找指定的控件，然后从控件中取得输入值。这种情况下就不用考虑 Eval()绑定和 Bind()绑定的区别了。代码如下：

```
protected void gvemployee_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    GridViewRow row = gvEmployee.Rows[e.RowIndex];
    string id = gvEmployee.DataKeys[e.RowIndex].Value.ToString();
    string title = ((DropDownList)row.FindControl("ddltitle")).SelectedValue;
    string notes = ((TextBox)row.FindControl("tbxnotes")).Text;
    doupdate(id, title, notes);
}
```

说明：使用 GridViewRow 对象的 FindControl 方法查找控件时，一定要确保方法参数的值与模板上该控件的 ID 属性值保持一致。

例 10-13 使用模板列实现多条记录的批量删除。

使用按钮列或命令列可以在 GridView 的每行显示一个按钮，单击按钮操作该行数据。有时需要对多行数据进行批量操作，这时可以使用模板列在每行显示一个复选框，用户勾选多个复选框后，再从 GridView 中获取所有选择项的主键，进行批量操作。

该示例的控件声明代码如下：

```
<asp:GridView ID = "gvEmployee" runat = "server" AutoGenerateColumns = "False"
    DataKeyNames = "employeeid">
    <Columns>
      <asp:BoundField DataField = "EmployeeID" HeaderText = "ID" ReadOnly = "True" />
      <asp:BoundField DataField = "FirstName" HeaderText = "First Name" />
      <asp:BoundField DataField = "LastName" HeaderText = "Last Name" />
      <asp:BoundField DataField = "Title" HeaderText = "Title" />
      <asp:TemplateField HeaderText = "选择">
          <ItemTemplate>
              <asp:CheckBox ID = "cbxSel" runat = "server" />
          </ItemTemplate>
          <ItemStyle HorizontalAlign = "Center" />
      </asp:TemplateField>
    </Columns>
</asp:GridView>
<asp:Button ID = "btndel" runat = "server" Text = "批量删除" onclick = "btndel_Click"
    OnClientClick = "return confirm('确认删除这些记录吗?');" />
```

运行效果如图 10-30 所示。

当单击“批量删除”按钮后，首先在客户端弹出确认删除对话框，用户选择“确定”后再触发服务器端的事件过程 btndel_Click，代码如下：

```
protected void btndel_Click(object sender, EventArgs e)
{
    List<int> ids = getselectedids ( ) ;
    if (ids.Count > 0)
```



图 10-30　使用模板列实现多条记录的批量删除

```
        {
            StringBuilder sb = new StringBuilder("Delete from Employees where employeeid in (" );
            foreach (int eid in ids)
            {
                sb.Append(eid); sb.Append(",");
            }
            sb[sb.Length - 1] = ')';
            delselected(sb.ToString( ) ) ;
        }
    }
```

该段代码首先调用 getselectedids 函数获取所有被选择员工的代码列表，然后使用 StringBuilder 对象动态构造一条批量删除数据的 SQL 语句，最后执行删除操作（这里省略代码）。

获取所有被选员工代码的函数如下：

```
    private List<int> getselectedids( )
    {
        List<int> list = new List<int>( ) ;
        foreach (GridViewRow row in gvEmployee.Rows)
        {
            if (row.RowType == DataControlRowType.DataRow)
            {
                CheckBox cbx = row.FindControl("cbxsel") as CheckBox ;
                if ( cbx != null && cbx.Checked )
                {
                    list.Add((int)gvEmployee.DataKeys[row.RowIndex].Value);
                }
            }
        }
```

```
        return list;
    }
```


这段代码中同样调用了 GridViewRow 对象的 FindControl 方法，在每一行中查找复选框对象，若该对象被勾选，再从主键列表中找到该行数据的主键，添加到 list 中返回。需要说明的是，在遍历所有表行时，必须要先判断该行是不是数据行，只有数据行才可以从中查找复选框控件，而其他行（Header、Footer 等）则略过。

10.3.2 ListView 控件

ListView 是 ASP.NET 3.5 及以上版本中新增的一个控件，用以取代 ASP.NET 1.X 中的 Repeater 控件。它是一个非常灵活的轻量级的控件，完全根据自定义的模板来呈现内容，并且提供了对选择、编辑等高级特性的支持。使用 ListView 最常见的场景是为了创建特殊的布局，如创建在一行中显示多条记录的表，或彻底脱离基于表格的呈现。

1. ListView 的模板

ListView 比 GridView 提供了更多的模板，如表 10-6 所示。

表 10-6 ListView 中可以使用的模板

列	描 述
ItemTemplate	设置所有数据项（没有使用 AlternatingItemTemplate 时）或奇数行数据项（使用 AlternatingItemTemplate 时）的内容和格式
AlternatingItemTemplate	和 ItemTemplate 配合，设置偶数行的内容和格式
ItemSeparatorTemplate	设置在项目中间绘制的分隔符的格式
SelectedItemTemplate	设置当前选定项目的模板
EditItemTemplate	设置数据项在编辑模式下的模板
InsertItemTemplate	设置插入新项目时使用的模板
LayoutTemplate	设置包装项目列表的布局模板
GroupTemplate	若使用了分组功能，则设置包装项目组的模板
GroupSeparatorTemplate	设置项目组的分隔符格式
EmptyDataTemplate	当绑定的数据对象为空时（没有记录或对象），使用该模板设置提示信息

在 ListView 呈现自身时，它首先对绑定的数据进行迭代，为每个数据行呈现 ItemTemplate；然后将所有的 Item 都放到 LayOutTemplate 里，从而实现布局控制。

2. 在 ListView 中呈现数据

在 Listview 中设置 ItemTemplate 的方法与 GridView 类似，所以这里主要的问题是如何将 Item 添加到整体布局中。请看如下示例。

例 10-14 使用 ListView 控件显示员工的信息。

ListView 控件的声明代码如下：

```
<asp:ListView ID = "lvEmployee" runat = "server" DataSourceID = "dsEmployee">
    <LayoutTemplate>
        < span id = "itemPlaceHolder" runat = "server"></span>
```

```
        </LayoutTemplate>
        <ItemTemplate><b>
            <%# Eval("EmployeeID") %> -
            <%# Eval("TitleOfCourtesy")%>
            <%# Eval("FirstName") %>
            <%# Eval("LastName") %>
            <hr /></b>
            <%# Eval("Address") %><br />
            <%# Eval("HomePhone") %><br />
            <%# Eval("Notes") %><br /><br />
        </ItemTemplate>
    </asp:ListView>
```

可以看出，ListView 中至少要定义两个模板：项目模板和布局模板。通过一个占位符将项目添加到布局中，这个占位符可以是各种各样的 HTML 元素，但其 ID 属性一定要用 itemPlaceHolder，并且 runat 属性必须为 Server。该页面的运行效果类似图 10-28。

3. 使用 GroupTemplate 分组显示数据

有时要在一行中显示多条数据，这就要通过分组模板来实现。

例 10-15 使用 ListView 显示员工的信息，每行显示三个员工。

ListView 控件的声明代码如下：

```
<asp:ListView ID="lvEmployee" runat="server" GroupItemCount="3"
      DataSourceID="dsEmployee" >
      <LayoutTemplate>
          <table border="0" cellpadding="10" width="100%">
                <tr id="groupPlaceholder" runat="server"></tr>
          </table>
      </LayoutTemplate>
      <GroupTemplate>
          <tr><td runat="server" id="itemPlaceholder" /></tr>
      </GroupTemplate>
      <ItemTemplate>
          <td valign="top"><b>
                <%# Eval("EmployeeID") %> -
                <%# Eval("TitleOfCourtesy")%>
                <%# Eval("FirstName") %>
                <%# Eval("LastName") %>
                </b><hr />
                <%# Eval("Address") %><br />
                <%# Eval("HomePhone") %><br />
                <%# Eval("Notes") %><br />
          </td>
      </ItemTemplate>
</asp:ListView>
```

该页面的运行效果如图 10-31 所示。

使用分组，就要先设置 ListView 的 GroupItemCount 属性，它决定每个组里项目的个数，本例中设置为 3。

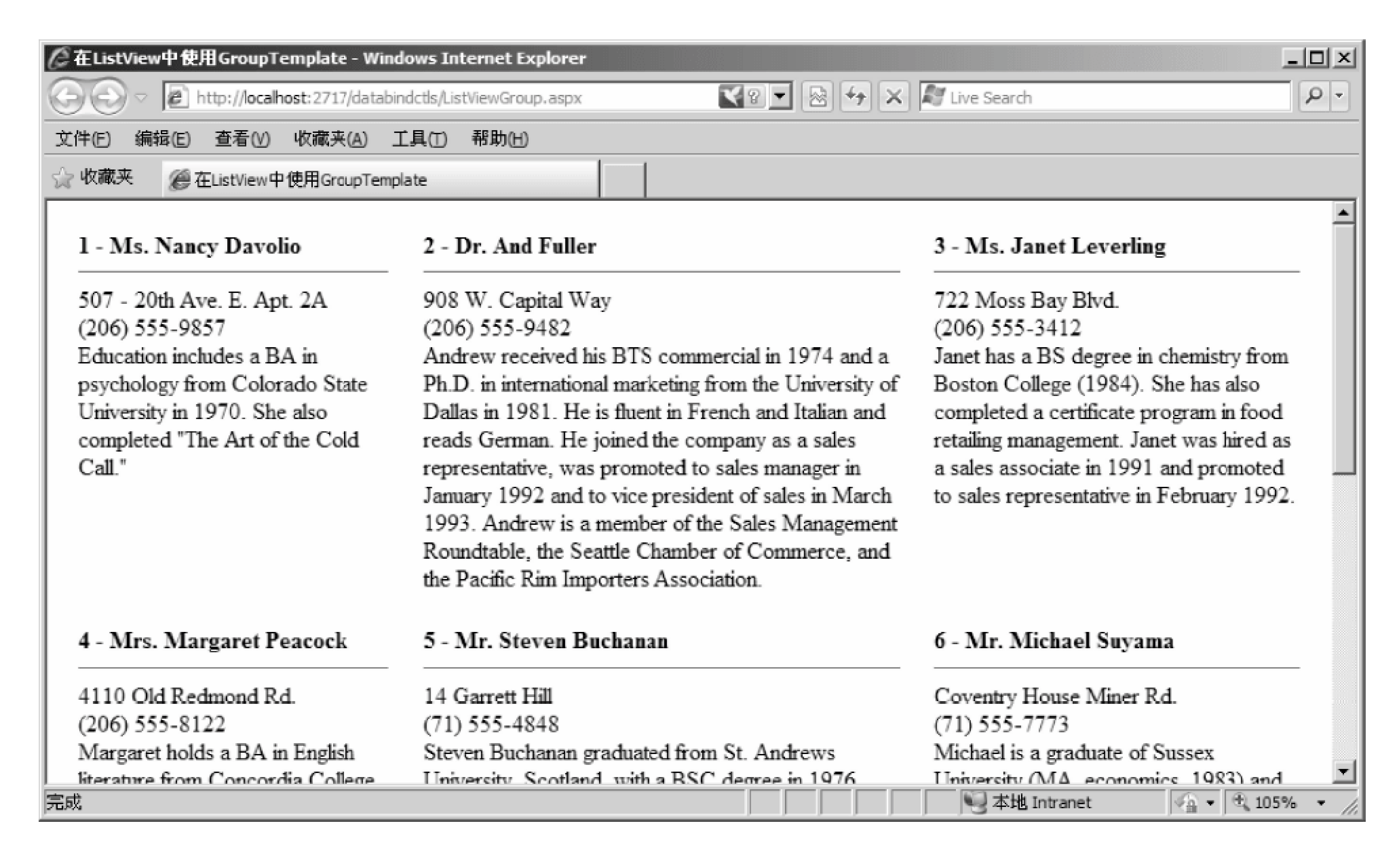


图 10-31　使用 ListView 的页面运行效果

有了组，就在数据和布局之间有了一个中间层，要将数据加入组，再将组加入布局中。这里同样使用了占位符，将名称为 groupPlaceholder 的占位符加入布局模板，再将名称为 itemPlaceholder 的占位符加入组模板，最后再定义 ItemTemplate。

关于 ListView 的高级特性，请参阅 MSDN 文档，这里不再详细介绍。

10.3.3　DetailsView 控件

GridView 和 ListView 都可以一次呈现多条记录，但有时会要求呈现单条记录的详细信息，这时可以使用 DetailsView 或 FormView。这两个控件都是每次显示一条记录，同时包含一个可选的分页按钮，用于在一组记录间导航；两者的区别在于 FormView 可以创建复杂的模板，使用更加灵活，DetailsView 使用简单的模板，相当于简版的 FormView。

DetailsView 使用表格布局的方式，每次显示一条记录，将每个数据项显示在表格的一行中。当 DetailsView 被绑定到一个项目集合上时，将显示集合中的第一个项目；若将 AllowPaging 属性设置为真，它还会自动创建一组分页控件，用于在各个项目之间导航。需要说明的是，使用 DetailsView 控件来浏览多条记录可能会存在效率问题，每当用户单击分页控件切换记录时，虽然最终只显示指定的一条记录，但系统却会获取所有满足条件的记录，会造成无谓的数据库负荷，实际应用中一定要考虑使用缓存或自定义分页的方式来优化程序。

1. 为 DetailsView 定义字段

当绑定到数据源时，DetailsView 可以使用反射的机制自动生成所有字段，也可以将它的 AutoGenerateColumns 属性设置为假，然后手工定义所有字段。可以像定义 GridView 的字段一样为 DetailsView 定义字段，只不过 GridView 的字段通常显示在列上，而 DetailsView 的字段通常显示在行上。

例 10-16 使用 DetailsView 控件开发一个员工信息浏览器。

页面的声明代码如下：

```
<asp:DetailsView ID="dvEmployee" runat="server" AutoGenerateRows="False"
     DataKeyNames="EmployeeID" DataSourceID="dsEmployee" AllowPaging="True">
     <Fields>
          <asp:BoundField DataField="EmployeeID" HeaderText="EmployeeID" />
          <asp:BoundField DataField="FirstName" HeaderText="FirstName" />
          <asp:BoundField DataField="LastName" HeaderText="LastName" />
          <asp:BoundField DataField="Address" HeaderText="Address" />
          <asp:BoundField DataField="HomePhone" HeaderText="HomePhone" />
          <asp:BoundField DataField="Notes" HeaderText="Notes" />
     </Fields>
 </asp:DetailsView>
 <asp:SqlDataSource ID="dsEmployee" runat="server"
     ConnectionString="<%$ ConnectionStrings:nwconnstr %>"
     SelectCommand="SELECTEmployeeID,FirstName,LastName,Address,HomePhone,
                    Notes FROM Employees">
</asp:SqlDataSource>
```

该页面的运行效果如图 10-32 所示。

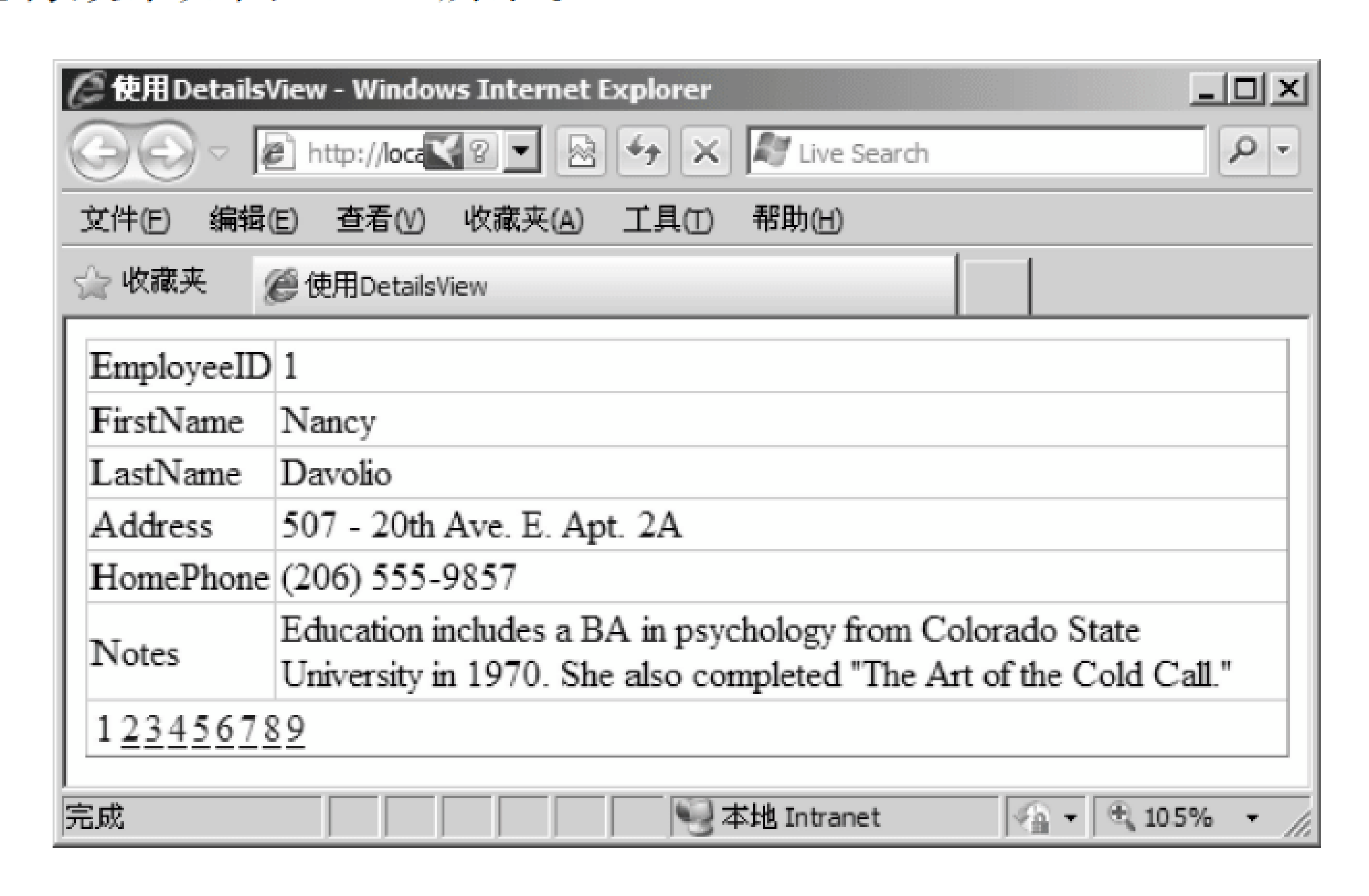


图 10-32 使用 DetailsView 控件的页面运行效果

在 DetailsView 中同样可以使用 HyperLinkField、ButtonField、CommandField、TemplateField 等，其使用方法及编程模型和 GridView 基本相同，这里不再过多说明。

2. 在 DetailsView 中进行记录操作

DetailsView 控件不仅能够浏览数据，还支持数据的插入、删除以及编辑操作。只要设置其 AutoGenerateInsertButton、AutoGenerateEditButton 及 AutoGenerateDeleteButton 属性值为真，系统就会自动在 DetailsView 的底部增加一行链接按钮，提供相应的功能。当单击“删除”按钮时，系统立刻执行删除操作；当单击“插入”或“编辑”按钮时，系统会进入编辑状态，允许添加新记录或修改当前记录。

DetailsView 共提供了三种操作模式，即只读模式、插入模式和编辑模式。默认 DetailsView 使用只读模式，只能浏览而不能更改数据；可以通过 CurrentMode 属性获取当

前的模式,并通过ChangMode方法改变当前模式。

在插入或编辑模式下,DetailsView使用标准的文本框来接收数据,如图10-33所示。在编辑模式下若切换记录,新显示的记录也处于编辑模式;若想改变这种行为方式,可以捕获PageIndexChanged事件,在其中调用ChangeMode方法将其改为只读模式。



图10-33 DetailsView用来接收数据的标准文本框

10.3.4 FormView控件

若想完全控制单条记录显示及编辑的样式,可以使用FormView控件,它完全依赖于模板,提供最大的灵活性。

同GridView类似,在FormView的模板列中可以使用ItemTemplate、EditItemTemplate、InsertItemTemplate、EmptyDataTemplate、HeaderTemplate、FooterTemplate及PagerTemplate等模板。

例10-17 使用FormView开发一个员工信息的增删改查程序。

为简单起见,本例中只处理EmployeeID、FirstName、LastName、TitleOfCourtesy和Notes等几个数据项。

(1) 首先建立两个数据源,声明代码如下:

```
<asp:SqlDataSource ID = "dsEmployee" runat = "server"
      ConnectionString = "<% $ ConnectionStrings:nwconnstr %>"
      SelectCommand = "SELECT EmployeeID, FirstName, LastName,TitleOfCourtesy,
                        Notes FROM Employees"
      UpdateCommand = "Update Employees set FirstName = @firstname,
           LastName = @lastname, TitleOfCourtesy = @titleofcourtesy, Notes = @notes
           Where EmployeeID = @employeeid"
      InsertCommand = "INSERT INTO Employees( LastName, FirstName, TitleOfCourtesy,
           Notes) VALUES (@lastname, @firstname, @titleofcourtesy, @notes)"
     DeleteCommand = "Delete From Employees Where EmployeeID = @employeeid">
</asp:SqlDataSource>
<asp:SqlDataSource ID = "dsTitle" runat = "server"
      ConnectionString = "<% $ ConnectionStrings:nwconnstr %>"
      SelectCommand = "SELECT distinct TitleOfCourtesy FROM Employees">
```

```
</asp:SqlDataSource>
```

第一个数据源提供了对 Employees 表的增删改查命令，第二个数据源获取称谓信息，以便在编辑模板中填充员工称谓下拉框。

(2) 向页面上添加一个 FormView 控件，设置其 DataSourceID 为刚才建立的 dsEmployee 数据源，并设置其 AllowPaging 属性为真，在其智能标记中单击"刷新架构"按钮。切换到源代码视图下，可以看到系统已经为 FormView 自动生成了 InsertTemplate、EditTemplate 及 ItemTemplate 模板。这时运行该页面，编辑状态下的效果如图 10-34 所示。

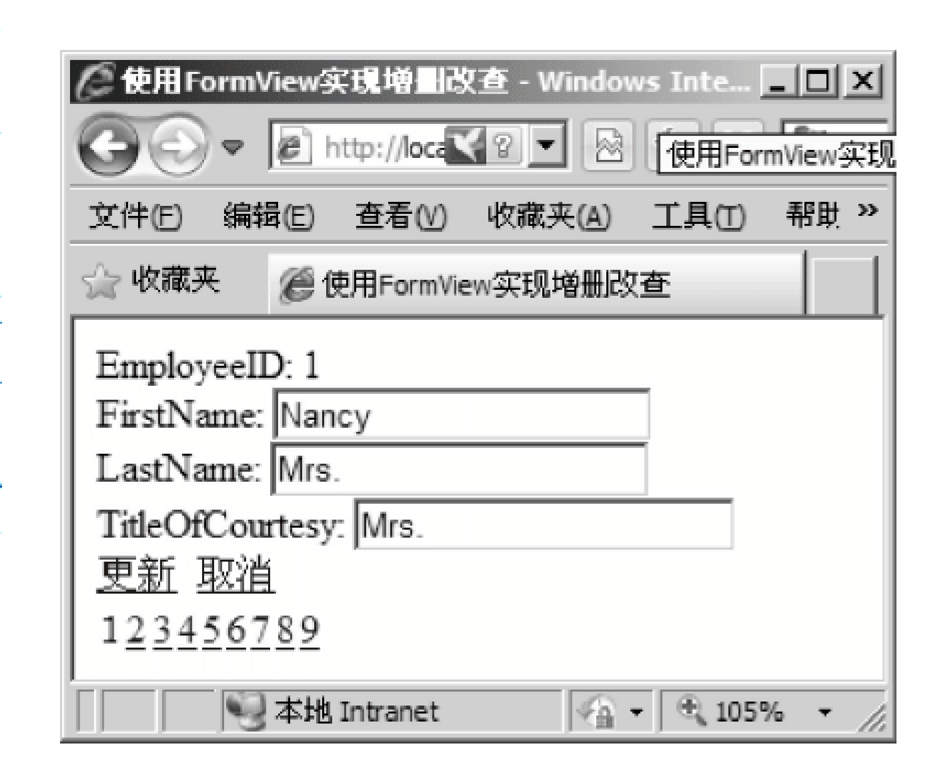


图 10-34 编辑状态下的页面效果

可以看到，系统自动为所有可编辑列创建 TextBox 以编辑数据。为避免输入错误的称谓，希望能够从下拉列表中选择 TitleOfCourtesy，需要对 EditItemTemplate 以及 InsertItemTemplate 进行定制。

(3) 定制 FormView 中的模板声明，详细如下：

```
<asp:FormView ID = "FormView1" runat = "server" DataKeyNames = "EmployeeID"
    DataSourceID = "dsEmployee" AllowPaging = "True">
    <EditItemTemplate>
        EmployeeID: <asp:Label ID = "EmployeeIDLabel1" runat = "server"
        Text = '<% # Eval("EmployeeID") %>' /><br />
        FirstName: <asp:TextBox ID = "FirstNameTextBox" runat = "server"
            Text = '<% # Bind("FirstName") %>' /><br />
        LastName: <asp:TextBox ID = "LastNameTextBox" runat = "server"
            Text = '<% # Bind("TitleOfCourtesy") %>' /><br />
        TitleOfCourtesy: <asp:DropDownList ID = "ddltitle" runat = "server"
            DataSourceID = "dsTitle" DataValueField = "TitleOfCourtesy"
            SelectedValue = '<% # Bind("TitleOfCourtesy") %>' /><br />
        <asp:LinkButton ID = "UpdateButton" runat = "server" CausesValidation = "True"
            CommandName = "Update" Text = "更新" />  
        <asp:LinkButton ID = "UpdateCancelButton" runat = "server"
            CausesValidation = "False" CommandName = "Cancel" Text = "取消" />
    </EditItemTemplate>
    <InsertItemTemplate>
        FirstName: <asp:TextBox ID = "FirstNameTextBox" runat = "server"
            Text = '<% # Bind("FirstName") %>' /><br />
        LastName: <asp:TextBox ID = "LastNameTextBox" runat = "server"
            Text = '<% # Bind("LastName") %>' /><br />
        TitleOfCourtesy: <asp:DropDownList ID = "ddltitle" runat = "server"
            DataSourceID = "dsTitle" DataValueField = "TitleOfCourtesy"
            SelectedValue = '<% # Bind("TitleOfCourtesy") %>' /><br />
        <asp:LinkButton ID = "InsertButton" runat = "server" CausesValidation = "True"
            CommandName = "Insert" Text = "插入" />  
```

```
        <asp:LinkButton ID="InsertCancelButton" runat="server"
            CausesValidation="False" CommandName="Cancel" Text="取消" />
    </InsertItemTemplate>
    <ItemTemplate>
        EmployeeID: <%# Eval("EmployeeID") %><br />
        FirstName: <%# Eval("FirstName") %><br />
        LastName: <%# Eval("LastName") %><br />
        TitleOfCourtesy: <%# Eval("TitleOfCourtesy") %><br />
        <asp:LinkButton ID="EditButton" runat="server" CausesValidation="False"
            CommandName="Edit" Text="编辑" /> 
        <asp:LinkButton ID="DeleteButton" runat="server" CausesValidation="False"
            CommandName="Delete" Text="删除" /> 
        <asp:LinkButton ID="NewButton" runat="server" CausesValidation="False"
            CommandName="New" Text="新建" />
    </ItemTemplate>
</asp:FormView>
```

可以看到，在编辑和插入模板中定义了 DropDownList 控件，其 DataSourceID 属性设置为 dsTitle，这样可以自动从数据库中获取所有称谓，其 SelectedValue 属性绑定到了数据项 TitleOfCourtesy 上，保证了保存数据时下拉框的选项值可以回传给 dsEmployee 数据源。更改后的编辑模板运行效果如图 10-35 所示。

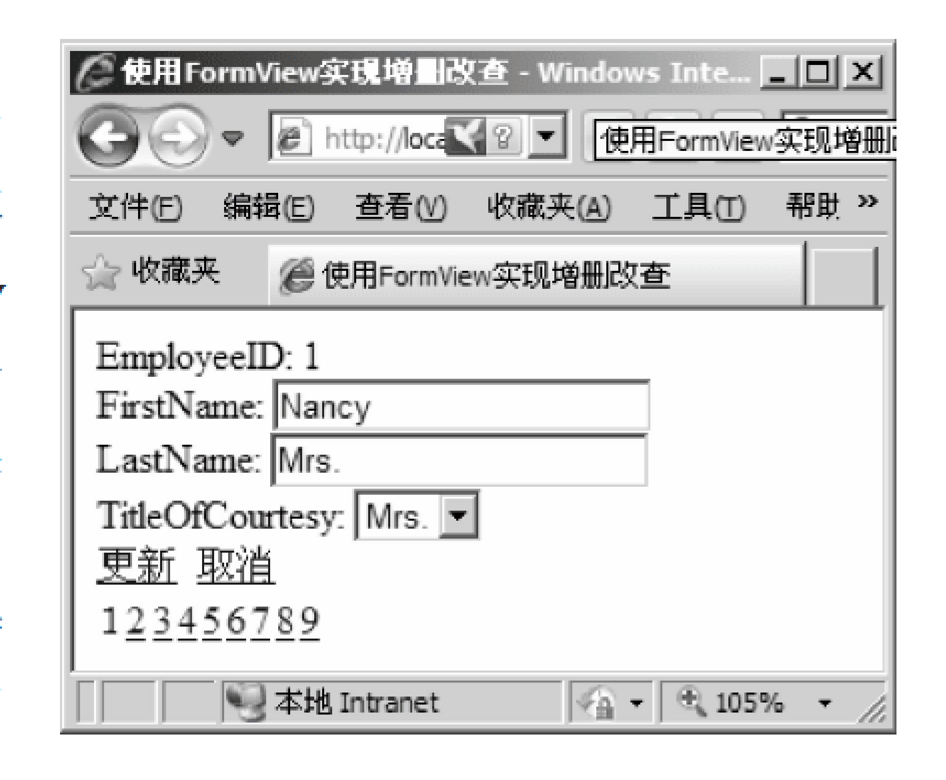


图 10-35 更改后的编辑模板运行效果

从示例中还可以看到，FormView 控件也支持只读、插入和编辑三种模式，也可在各种模式之间切换。但与 GridView 及 DetailsView 不同的是，FormView 不支持自动创建按钮列(CommandField)，必须手工创建各种按钮对象。注意，在 ItemTemplate 中创建了编辑、删除和新建三个按钮，在编辑模板中创建了更新和取消两个按钮，在插入模板中创建了插入和取消两个按钮。可以使用 Button 或 LinkButton 创建按钮，所有按钮的 CommandName 属性必须设置为合适的值，这样才能触发相应的事件，自动进行模式切换。常用的命令名如表 10-7 所示。

表 10-7 FormView 的命令按钮中可以使用的 CommandName 值

命令	作用
Edit	适用于 ItemTemplate，从只读模式切换到编辑模式以编辑当前项
Cancel	适用于 EditItemTemplate 和 InsertItemTemplate，在编辑或插入模式下放弃数据，返回到只读模式
Update	适用于 EditItemTemplate，将编辑后的数据保存下来，并返回只读模式
New	适用于 ItemTemplate，插入一条新数据并转入编辑模式
Insert	适用于 InsertItemTemplate，将插入的数据保存下来，并返回只读模式
Delete	适用于 ItemTemplate，直接删除当前项

10.4　习题与上机练习

1. 选择题

(1) 为将页面的 PageNum 属性值绑定到某 Label 控件上显示，需设置该控件的 Text 属性为(　　)数据绑定表达式。

A. ＜％＃ PageNum ％＞　　B. ＜％＄ PageNum ％＞

C. ＜％＃ Eval("PageNum") ％＞　　D. ＜％＃ Bind("PageNum") ％＞

(2) 在绑定了数据源的 Repeater 对象中，系统会自动提供(　　)对象，可以使用该对象的 Eval 方法从指定的列中检索数据。

A. Container　　B. DataBinder

C. DataReader　　D. DataTable

(3) 在 DataList 控件中，对任何一个按钮单击时，都会触发(　　)事件。

A. EditCommand　　B. ItemCommand

C. CancelCommand　　D. SelectCommand

(4) 在使用 DataView 对象进行筛选和排序等操作之前，必须指定一个(　　)对象作为 DataView 对象的数据来源。

A. DataTable　　B. DataGrid

C. DataRows　　D. DataSet

(5) 在包含多个表的 DataTable 对象的 DataSet 中，可以使用(　　)对象来使一个表和另一个表相关联。

A. DataRelation　　B. Collections

C. DataColumn　　D. DataRows

(6) 在 GridView 控件中设定显示学生的学号，姓名，出生日期等字段。现要将出生日期设定为短日期格式，则应将数据格式表达式设定为(　　)。

A. {0:d}　　B. {0:c}

C. {0:yy-mm-dd}　　D. {0:p}

(7) XMLDateSource 与 SiteMapDataSource 数据源控件能够用来访问(　　)。

A. 关系型数据　　B. 层次性数据

C. 字符串数据　　D. 数值型数据

2. 简答题

(1) 试举例说明如何计算一个表达式的值，并将其绑定到 Label 控件上显示出来。

(2) 构造一个课程的集合，每门课程包含课程号、课程名、学时等数据项；在页面上添加一个课程列表框，能够显示集合中所有课程的名称，但提交所选的课程号。请写出代码。

(3) ASP.NET 2.0 的数据源控件起什么作用?

(4) ASP.NET 2.0 共提供了哪几种类型的数据源控件？分别在什么场合下使用?

(5) 在 SqlDataSource 配置中使用命令参数时，可采用哪几种方式获得参数值？举例说明。

(6) 在访问关系数据库的应用中，使用 SqlDataSource 有什么局限性？为什么要使用

ObjectDataSource?

(7) 试比较 GridView、DetailView、FormView 和 ListView 四种控件的特点,并分别说明每种控件的用途。

(8) 使用 GridView 控件显示大量数据时为什么要采用自定义分页? 请深入分析其原因,并举例说明如何实现自定义分页控制。

(9) 在处理 GridView 的行数据时,通常可以使用哪些行命令? 分别会触发哪些行事件?

(10) GridView 的模板列中,可以使用哪些常用的模板类型? 分别代表什么?

(11) 在模板列中,使用 Eval()和 Bind()方法都可以绑定到数据项,试比较两种用法的异同。

3. 上机练习

接第 9 章的上机练习题。

以管理员身份登录系统后,进入用户信息管理界面,实现用户信息的编辑(删除、编辑、查询)。用 GridView 数据绑定控件实现。

第 11 章　Web Service 技术

网络发展到今天，没有一个概念能像 Web Service 这么快地流行起来并引起广泛关注的了。特别是越来越多的商业机构希望把它们的企业运营集成到分布式应用软件环境中，如网上支付、网上购物、网上订票和网上炒股等就需要一个基于 Internet 开发标准的分布计算模式。这种分步计算模式必须独立于提供商、平台和编程语言；同时，它必须易于实现和发布应用程序。这就使得 Web Service 技术应运而生了。

11.1　Web Service 的概念

11.1.1　Web Service 的定义和概念

关于 Web Service，有很多种不同的概念和定义，它们都是站在不同的角度给出的。下面分别列出一些概念和定义，以帮助读者深入理解 Web Service。

(1) Web Service 是一种跨编程语言和跨操作系统平台的远程调用技术。

所谓远程调用，就是计算机 A 上的程序可以调用另一台计算机 B 上的某个对象的方法。例如，银联提供给商场的 POS 刷卡系统。还有，亚马逊网站 amazon. com、天气预报系统、淘宝网、校内网、百度等，它们把自己的系统服务以 Web Service 服务的形式暴露出来，让第三方网站和程序可以调用这些服务功能，这样就扩展了自己系统的市场占有率。

所谓跨编程语言和跨操作平台，就是说服务端程序采用 java 编写，客户端程序采用其他编程语言；反之亦然，跨操作系统平台则是指服务端程序和客户端程序可以在不同的操作系统上运行。

(2) Web Service 是一个专用的软件系统，其目的是支持跨网络的计算机间互操作。这就是说，Web Service 可以看做 Web 应用程序的编程接口。

具体来说，网站可看做向人们提供各种信息服务的聚合体，网站的服务通过 Web Service 来表达。每个网站都可以将它向外界所提供的服务封装为一个或多个 Web Service，其他网站或 Web 程序可以通过调用这些 Web Service 来使用此网站所提供的功能。

例如，某天气预报网站需要为人们提供各地的天气情况，但并不需要自己维护一个庞大的全球气象数据库。因为，如果世界各地的气象局网站都以 Web Service 方式提供实时的气象信息查询服务，那么该网站就可以根据目的地有针对性地查询特定地区的气象局网站，然后将收到的气象信息加工后返回给客户端浏览器。

(3) Web Service 是松散耦合、可复用的软件模块，从语义上看，它封装了离散的功能，在 Internet 上发布后能够通过标准的 Internet 协议在程序中访问。

第一，Web Service 是可复用的软件模块。基于组件的模型允许开发者复用他人创建的代码模块，组成或扩展它们，形成新的软件。

第二，这些软件模块是松散耦合的。传统的软件设计模式要求各单元之间紧密连接，这种连接一旦建立，很难把其中一个元素取出并用另一个元素代替；相反，松散耦合的系统，只需要很简单的协调，并允许更加自由的配置。

第三，Web Service 封装了离散的功能。一个 Web Service 就是一个自包含的"小程序"，完成单个任务。Web Service 的模块使用其他软件可以理解的方式描述输入和输出，其他软件知道它能做什么，如何调用它以及返回什么样的结果。

第四，Web Service 可以在程序中访问。Web Service 不需要图形化的用户界面，它在代码级工作，可以被其他软件调用并交换数据。

第五，Web Service 是在 Internet 上发布的，使用 HTTP 协议。

(4) Web Service 有时被称为 XML Web Service，允许通过一个应用程序向另一个应用程序发送 XML 消息，实现数据交换和方法的远程调用。Web Service 使用多种标准，使得大多数应用程序都可以读这些消息。

11.1.2 Web Service 的基本特征

Web Service 实际上是一种应用程序，使用 HTTP 协议在网上提供函数接口，能够让不同的应用程序之间进行交互操作，即用户可以从任何地方调用 Web Service。

概括起来，Web Service 具有以下特征：

(1) Web Service 是分布式的、可复用的 Web 应用程序组件。通过聚合已有的 Web Service 可以快速开发能够提供各种信息与服务的 Web 应用程序，而这些应用程序自身也可以发布新的 Web Service 以供其他程序调用。

(2) 支持松散耦合。应用程序与 Web Service 执行交互前，它们之间的连接是断开的。当应用程序需要调用 Web Service 的方法时，两者之间建立了连接。当实现了相应功能后，两者之间的连接断开。应用程序与 Web 应用之间的连接是动态建立的，实现了系统的松耦合。

(3) 跨平台性。Web Service 使用 XML 表达信息，使用 HTTP 协议传输信息。对于不同的平台，只要能够支持编写和解释 XML 文件就能够实现不同平台之间应用程序的相互通信。

(4) 自描述性。Web Service 使用 WSDL 作为自身的描述语言，而 WSDL 是基于 XML 格式的纯文本描述，包含了被 Web 应用程序调用所需的全部信息，WSDL 能够帮助其他 Web 应用程序访问 Web Service。

(5) 可发现性。应用程序可以通过 Web Service 提供的注册中心查找和定位 Web Service。

11.1.3 Web Service 的优势

Web Service 的主要目标是跨平台的可互操作性。在以下 4 个方面，Web Service 具有极大的优势。

1. 跨防火墙的通信

如果应用程序有成千上万、分布在世界各地的用户，那么客户端和服务器之间必然有防火墙或代理服务器。在这种情况下，一个用户界面和中间层有较多交互的应用程序，使用 Web Service 可以节省用于开发用户界面的 20%时间。由 Web Service 组成的中间层，也可以在应用程序集成或其他场合下重用。

2. 应用程序集成

在企业级的应用程序中，经常要把用不同语言、在不同平台上运行的各种程序集成起来，这将花费很大的开发力量。通过 Web Service，应用程序可以用标准的方法把功能和数据"暴露"出来，供其他应用程序使用。

3. B2B 的集成

跨公司的商务交易集成通常叫做 B2B 集成。只要把商务逻辑"暴露"出来，成为 Web Service，就可以让任何指定的合作伙伴调用这些商务逻辑，而不管他们的系统使用何种平台和开发语言。这样就大大减少了花在 B2B 集成上的时间和成本，使许多原本无法承受 EDI 的中小企业也能实现 B2B 集成。

4. 软件和数据重用

Web Service 在允许重用代码的同时，也可以重用数据。使用 Web Service，再也不必像以前那样，要购买第三方组件才能安装、调用，而是直接调用远端的 Web Service 就可以了。另一种重用情况，是集成几个应用程序的功能。现在 Web 上有很多应用程序供应商，把他们实现的功能通过 Web Service"暴露"出来，这样使用者就可以将它们集成到自己的门户站点中，为访问用户提供一个统一、友好的界面。

当然，也有一些情况，使用 Web Service 不能带来任何好处。例如，单机应用程序，以及同一个局域网上的同构应用程序等。

11.2 Web Service 的实现技术

11.2.1 Web Service 的体系结构

Web Service 包括 4 个组成部分，分别是 Web 服务(Web Service 自身)、服务提供者(Service Provider)、服务请求者(Service Requester)和服务的注册中心(Service Registry)。通常，将服务提供者、服务请求者和服务的注册中心称为 Web Service 的三大角色。这三大角色及其行为共同构成了 Web Service 的体系结构，如图 11-1 所示。

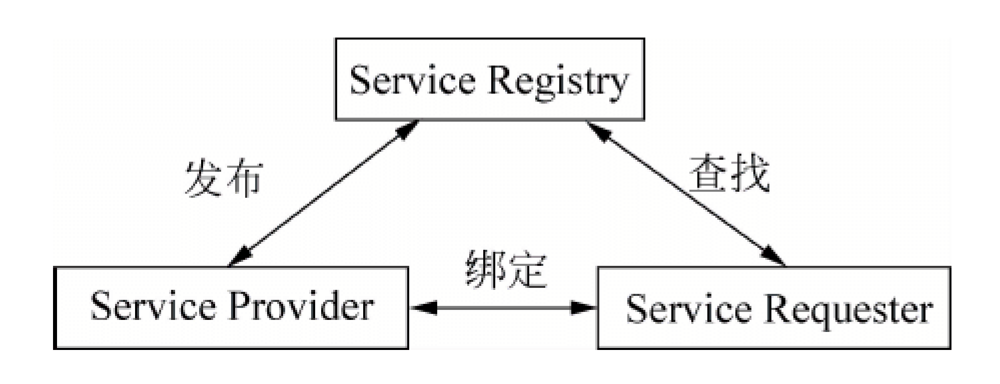


图 11-1 Web Service 的体系结构

服务提供者也称为服务的拥有者，通过提供服务接口使 Web Service 在网络上是可用的。服务接口是可以被其他应用程序访问和使用的软件组件，如果服务提供者创建了服务

接口，服务提供者会向服务注册中心发布服务，以注册服务描述。相对于 Web Service 而言，服务提供者可以看做访问服务的托管平台。

服务请求者也称为 Web Service 的使用者，服务请求者通过注册中心查找需要的服务，当请求者通过注册中心查找到服务的提供者之后，就会绑定到服务接口上，与服务提供者进行通信。相对于 Web Service 而言，服务请求者是寻找和调用提供者提供的服务接口的应用程序。

服务注册中心可以使请求者和提供者进行信息通信，当服务提供者提供服务接口后，注册中心则会接收提供者发出的请求，从而注册提供者。服务请求者对注册中心进行服务请求后，注册中心能够查找到提供者并绑定到请求者。

那么，具体地说，Web Service 的运转，如图 11-2 所示，它主要基于以下方面：

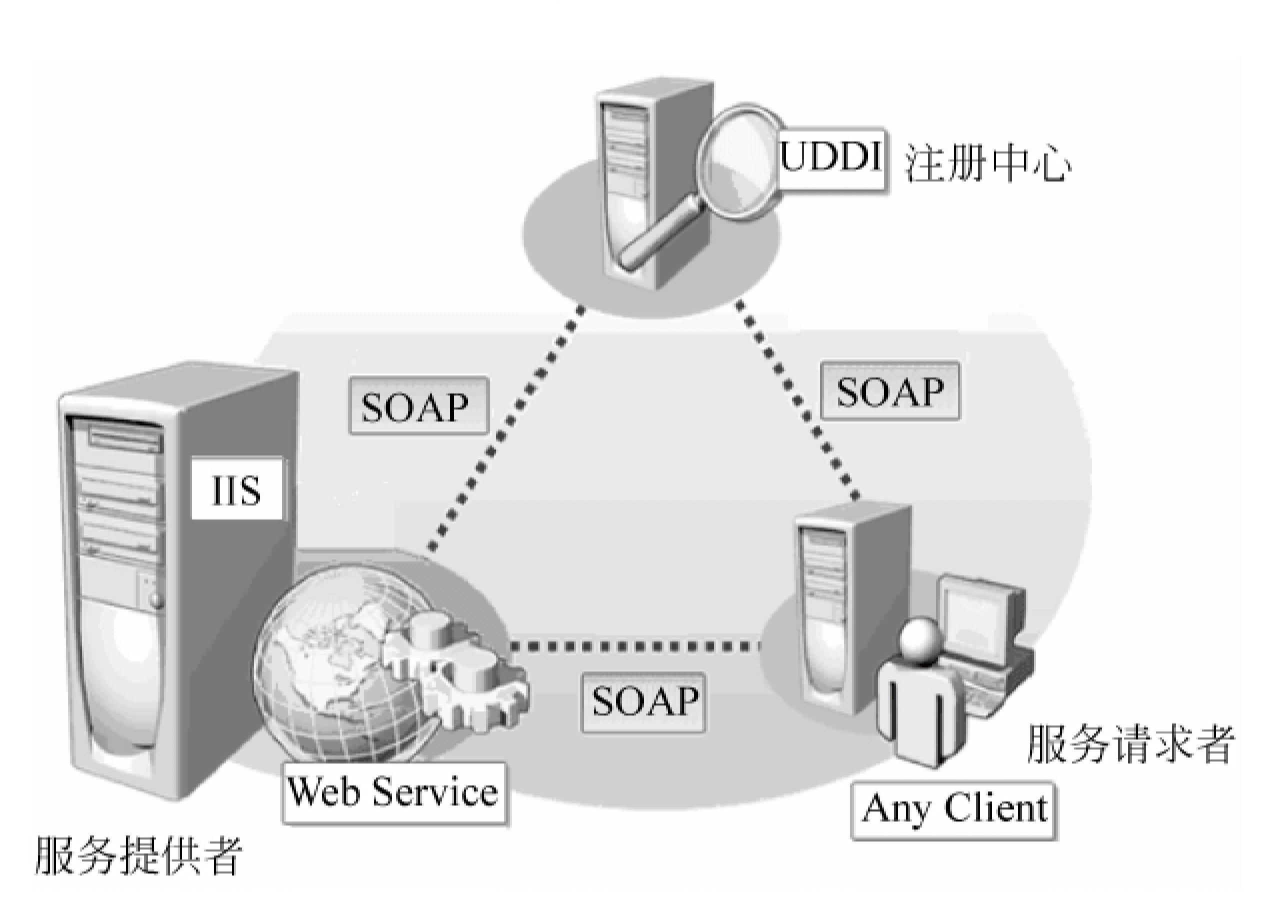


图 11-2　Web Service 的运转

(1) Web Service 驻留于 Web Server 中。

(2) 使用 UDDI 机制查找符合要求的 Web Service。

(3) 网络中的计算机通过 SOAP 协议进行通信。

11.2.2　Web Service 的协议栈

在 Web Service 体系结构中，为了保证体系结构中的每个角色都能够正确执行发布、查找和绑定操作，Web Service 体系必须为每一层标准技术提供 Web Service 协议栈。Web Service 协议栈如图 11-3 所示。上层的功能必须依靠下层的支持。

在 Web Service 的技术架构中，最底层是网络传输层。网络传输层可以使用多种协议，包括 HTTP、FTP 以及 SMTP 等。

在网络传输层的上一层则是消息传递层。消息传递层使用 SOAP 作为消息传递协议，以实现服务提供者、服务注册中心和服务请求者之间的信息交换。

在消息传递层之上的是服务描述层。服务描述层使用 WSDL 作为消息协议，WSDL 使用 XML 语言来描述网络服务，能够提供 Web Service 的一些特定信息。服务描述层包括了 WSDL 文档，这些文档包括功能、接口、结构等定义和描述。

在服务描述层之上的是服务发布层，该层使用 UDDI 协议作为服务的发布/集成协议。

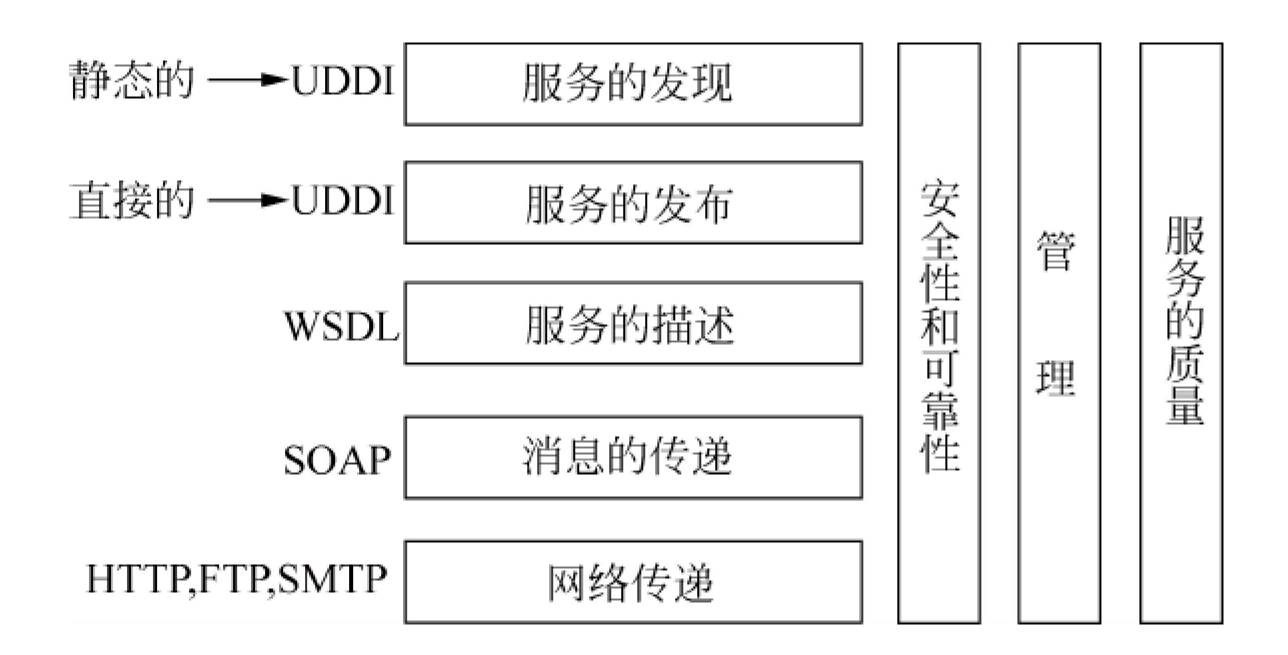


图 11-3 Web Service 的技术架构

服务发布层能够为提供者提供发布 Web Service 的功能,也能够为服务请求者提供查询、绑定的功能。

11.2.3 Web Service 的核心元素

简单地说,Web Service 的定义:通过 SOAP 在 Web 上提供的软件服务,使用 WSDL 文件进行说明,并通过 UDDI 进行注册。

下面分别对这几种主要技术进行详细说明。

1. SOAP 协议

简单对象访问协议(Simple Object Access Protocol,SOAP)是 Web Service 的传输协议。Web Service 的提供者与使用者之间的信息交换必须遵循 SOAP 协议,它规定了 Web Service 的提供者与使用者之间信息的编码和传送方式。

SOAP 协议是基于 HTTP 协议的互联网应用层协议,允许信息穿过防火墙而不被拦截。两者的关系就好比高速公路(SOAP)是基于普通公路(HTTP)改造的,在一条普通公路上加上隔离栏后就成了高速公路。这个隔离栏就是 XML。因此,SOAP 是利用 XML 来封装消息的,它满足下述关系:

```
SOAP 协议 = HTTP 协议 + XML 格式
```

SOAP 仅是一种约定,是平台中立与语言无关的。因此,Web Service 使用 SOAP 协议实现了跨编程语言和跨操作系统平台。

由于 Web Service 采用 HTTP 协议传输数据,采用 XML 格式封装数据。各种编程语言对 HTTP 协议和 XML 技术都提供了很好的支持,Web Service 客户端与服务器端使用任意编程语言都可以完成 SOAP 的功能,所以,Web Service 很容易实现跨编程语言,跨编程语言自然也就跨了操作系统平台。

2. WSDL 语言

WSDL(Web Service Description Language)是一种纯文本形式的 Web Service 描述语言,基于 XML 规范而制定。

WSDL 用于动态发布 Web 服务、查找已发布的 Web 服务、绑定 Web 服务。

WSDL 文件本身是一个 XML 文档,用于描述 Web Service 的详细信息,即说明一组 SOAP 消息以及如何交换这些消息。它与编程语言无关,可以从不同平台、以不同编程语言



访问 Web Service 接口。

WSDL 文件保存在 Web 服务器上，通过一个 URL 地址可以访问到它。客户端调用一个 Web Service 之前，需要知道该地址。Web Service 提供商通过两种方式发布它的 WSDL 文件地址。

- 注册到 UDDI 服务器，以便被人查找；
- 直接告诉给客户端调用者，如在自己的网站发布。

3. UDDI 机制

一旦客户端能够获得 WSDL 文档，就应该有足够的信息知道如何与目标 Web Service 进行交互。如果客户端知道 WSDL 文档驻留的位置，那么只需通过 HTTP 就能请求它。然而，如果客户端不知道 WSDL 驻留在哪里，就需要一个发现机制。

统一描述、发现和集成（Universal Description ，Discovery and Integration，UDDI）是一种用于查找 Web Service 的机制。

UDDI 服务器存储了 Web Service 的相关信息，也就是此 Web Service 的 WSDL 文档，可供 Web 应用程序来定位和引用 Web Service。如图 11-1 所示，Web Service 的使用者先通过 UDDI 查找所需的 Web Service，获取其 WSDL 文档，然后根据此 WSDL 文档就可以定位并引用 Web service。在整个过程中，UDDL 起到了一个类似于"电话号码簿"的作用。

因此，UDDI 的意图就是作为一个注册簿，企业可以在不同的目录下注册它们自己或其服务。通过浏览一个 UDDI 注册簿，用户能够查找一种服务或一个公司，并发现如何调用该服务。

UDDI 的目录条目分为三类：

- 白页：企业实体的列表，介绍提供服务的公司和企业，包括名称、地址、联系方式和已知标识符等。
- 黄页：包括基于标准分类法划分的行业类别信息。
- 绿页：是调用一项服务所必需的文档，详细介绍了访问服务的接口，以便用户能够编写应用程序以使用 Web Service。

11.3 构建 ASP.NET Web Service

使用 Visual Studio 可以很方便地构建 Web Service，本节将举例说明 Web Service 的创建和发布方法。

11.3.1 使用 Visual Studio 创建 Web Service

启动 Visual Studio 2010，在"文件"菜单中选择"新建网站"命令，在打开的"新建网站"对话框中，首先要选择.NET Framework 3.5 版本（默认的 4 版本没有 Web 服务），然后选择"ASP.NET Web 服务"，如图 11-4 所示。单击"确定"按钮，系统将自动创建一个名为 Service. cs 的文件，在其中定义一个名为 Service 的 Web Service，并自动生成一个公有方法 Hello World，显示如下代码。

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Web;
using System.Web.Services;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
//若要允许使用 ASP.NET AJAX 从脚本中调用此 Web 服务,请取消对下行的注释
// [System.Web.Script.Services.ScriptService]

public class Service : System.Web.Services.WebService
{
    public Service ()
    {
        //如果使用设计的组件,请取消注释以下行
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld()
    {
        return "Hello World";
    }
}
```



图 11-4　创建 ASP.NET Web 服务

上述代码中,Service 类从 WebService 类派生,包括两个公有方法：Service ( ) 和 HelloWorld()。其中,[WebMethod] 标签声明了一个 Web 方法,通知 ASP.NET 编译器,接下来的 Web 方法可以通过互联网调用。

一个 Web Service 可以拥有多个公有的 Web 方法。这些 Web 方法不能返回对象,只能

返回一个值类型数据(字符串数组、字符串、数值等)。例如,HelloWorld()返回一个字符串。

上述代码位于 App_Code/Service.cs 文件中,可以根据需要继续添加 Web 方法。例如,在 HelloWorld()方法后面添加一个 Add()方法,代码如下。

```
[WebMethod]
public int Add(int x, int y)
{
    return (x + y);
}
```

与 Service.cs 文件相对应,Service.asmx 文件本身只有一行代码:

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs" Class="Service" %>
```

这里,页面指令“@ WebService”指明本页定义的是一个 Web Service,实现该 Web Service 的类是 Service,位于“~/App_Code/Service.cs”文件中。

11.3.2 测试 Web Service

从“调试”菜单选择“启动调试”命令,浏览器会自动启动以测试 Web Service 是否工作正常。如图 11-5 所示,可以看到测试网页中列出了 Service 类里的所有 Web 方法。



图 11-5　Web Service 测试(1)

单击 Add 方法，Web Service 应用程序会跳转到另一个页面，如图 11-6 所示，该页面提供了方法的调用测试。在文本框中分别输入两个数字，单击“调用”按钮，则取回 Web Service 的计算结果。该结果以 XML 格式被返回，如图 11-7 所示。

Service Web 服务 - Windows Internet Explorer

http://localhost:1875/W

百度一下，你就知道

文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)

收藏夹 Service Web 服务

Service

单击此处，获取完整的操作列表。

Add

测试

若要使用 HTTP POST 协议对操作进行测试，请单击“调用”按钮。

参数	值
x:	12
y:	5

调用

SOAP 1.1

以下是 SOAP 1.2 请求和响应示例。所显示的**占位符**需替换为实际值。

```
POST /WebSite4/Service.asmx HTTP/1.1
Host: localhost
```

Internet 100%

图 11-6 Web Service 测试(2)

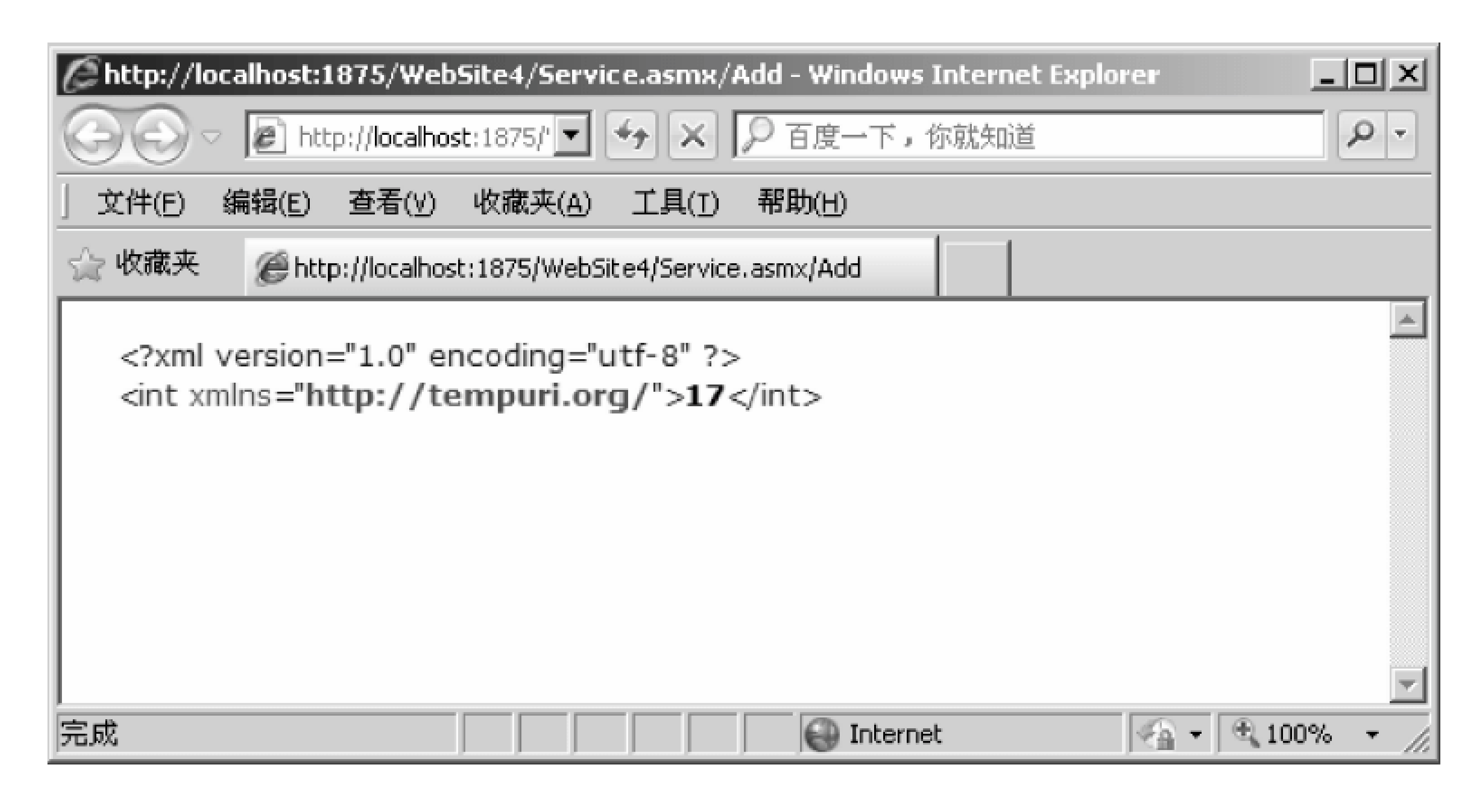


图 11-7 Web Service 的返回结果

至此，完成了一个简单的 Web Service 开发工作。

11.3.3 发布 Web Service

Web Service 本身是不能单独存在的，必须依赖于一个网站进行发布。因此，发布 Web Service，就像发布一个普通的 ASP.NET 网站一样，通过在 IIS 上设置虚拟目录、发布提供有 Web Service 的网站到 IIS 上。

假设本节的 Web Service 网站，被发布到本地 IIS 的 D:\Website4 应用程序文件夹，对该文件夹创建了一个名为 AddService 的虚拟目录，如图 11-8 和图 11-9 所示。

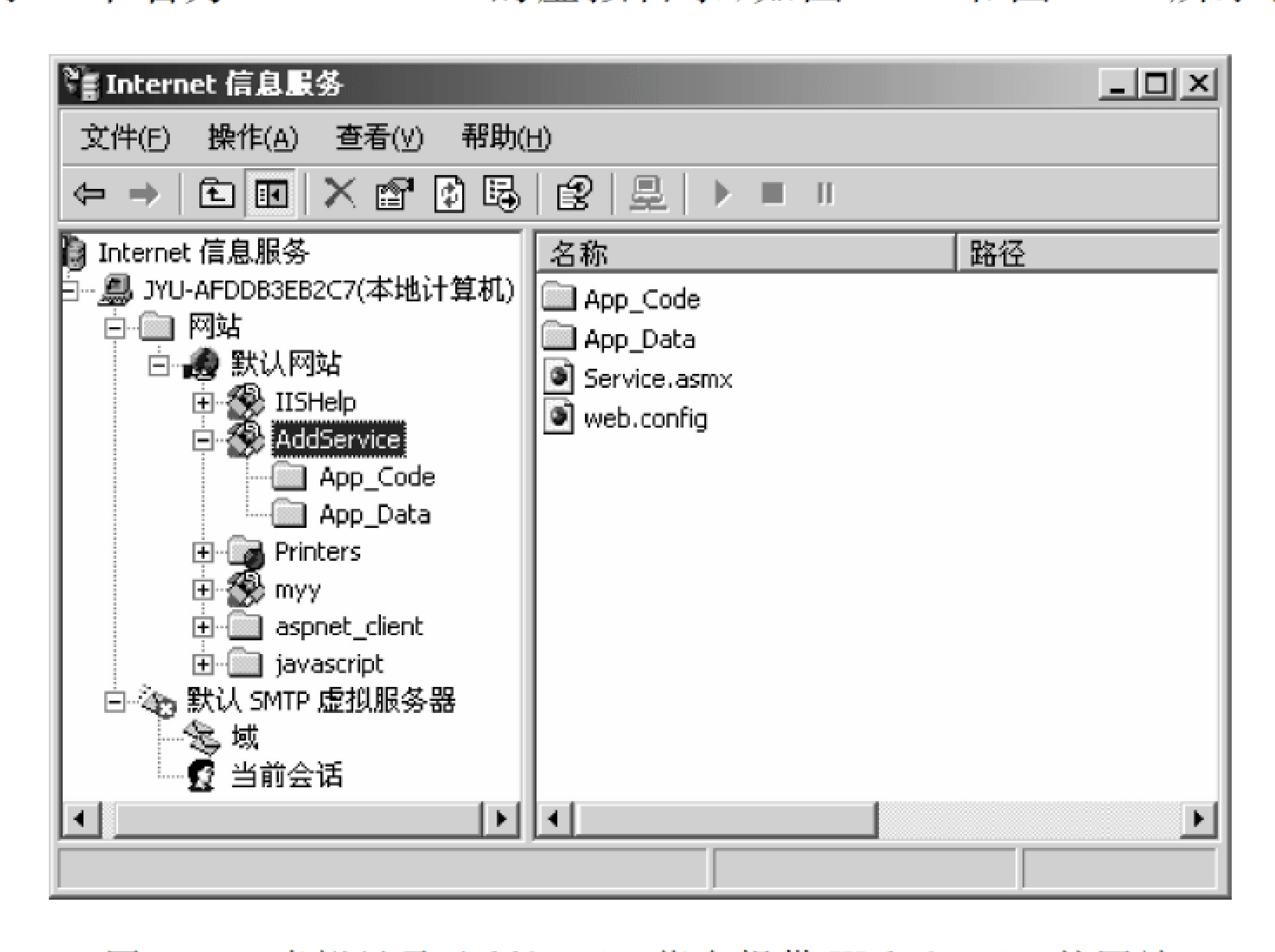


图 11-8 虚拟目录 AddService 指向提供 Web Service 的网站



图 11-9 打开网站

由于 Web Service 的源文件扩展名为 asmx，发布之后的 Web Service 可以通过 http://localhost/AddService/Service.asmx 进行访问。

11.4 使用 Web Service

使用 Web Service 的过程，实际上是 Web Service 的使用者与 Web Service 实现绑定，并调用其方法的过程。绑定有两种方式：一种是动态的；另一种是静态的。动态通过 UDDI 实现，静态通过已经获得的 WSDL 文件实现。这里使用静态绑定，即使用 Visual Studio 在 ASP.NET 应用程序中引用已经开发好的 Web Service。

11.4.1 添加 Web 引用

在 ASP.NET 网站中，使用 Web Service 的方法是从“网站”菜单中选择“添加 Web 引用”命令，则出现如图 11-10 所示的页面。

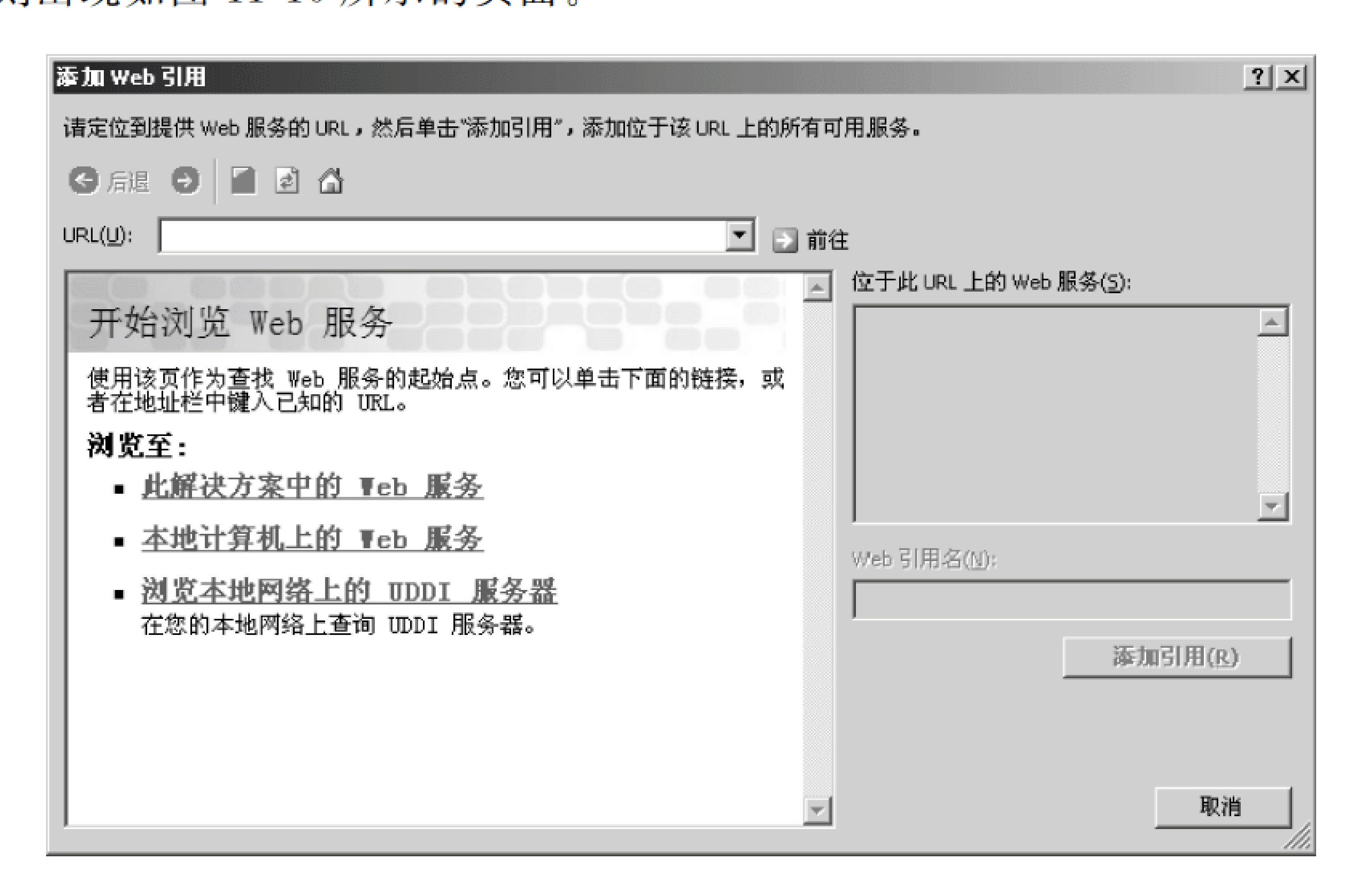


图 11-10 “添加 Web 引用”对话框(1)

由于本节中的 Web Service 已经在本地 IIS 上发布，所以单击“本地计算机上的 Web 服务”项，进行相应选择后，进入如图 11-11 所示的页面。此时 Web 引用的默认名为 localhost，单击“添加引用”按钮，即可完成向项目中添加 Web 引用的工作，如图 11-12 所示。

在图 11-12 中，可以看到 Visual Studio 为添加 Web 引用的网站自动生成了三个调用 Web Service 的相关文件。其中，Server.disco 文件用于指明如何获取 Web Service 的相关信息，Service.wsdl 是 Web Service 的描述文件，Server.discomap 文件用于指明上述两个文件的 URL 地址。

11.4.2 访问 Web Service

添加了 Web 引用之后，ASP.NET 网站使用以下代码访问 Web Service。

```
protected void btnAdd_Click(object sender, EventArgs e)
{
```

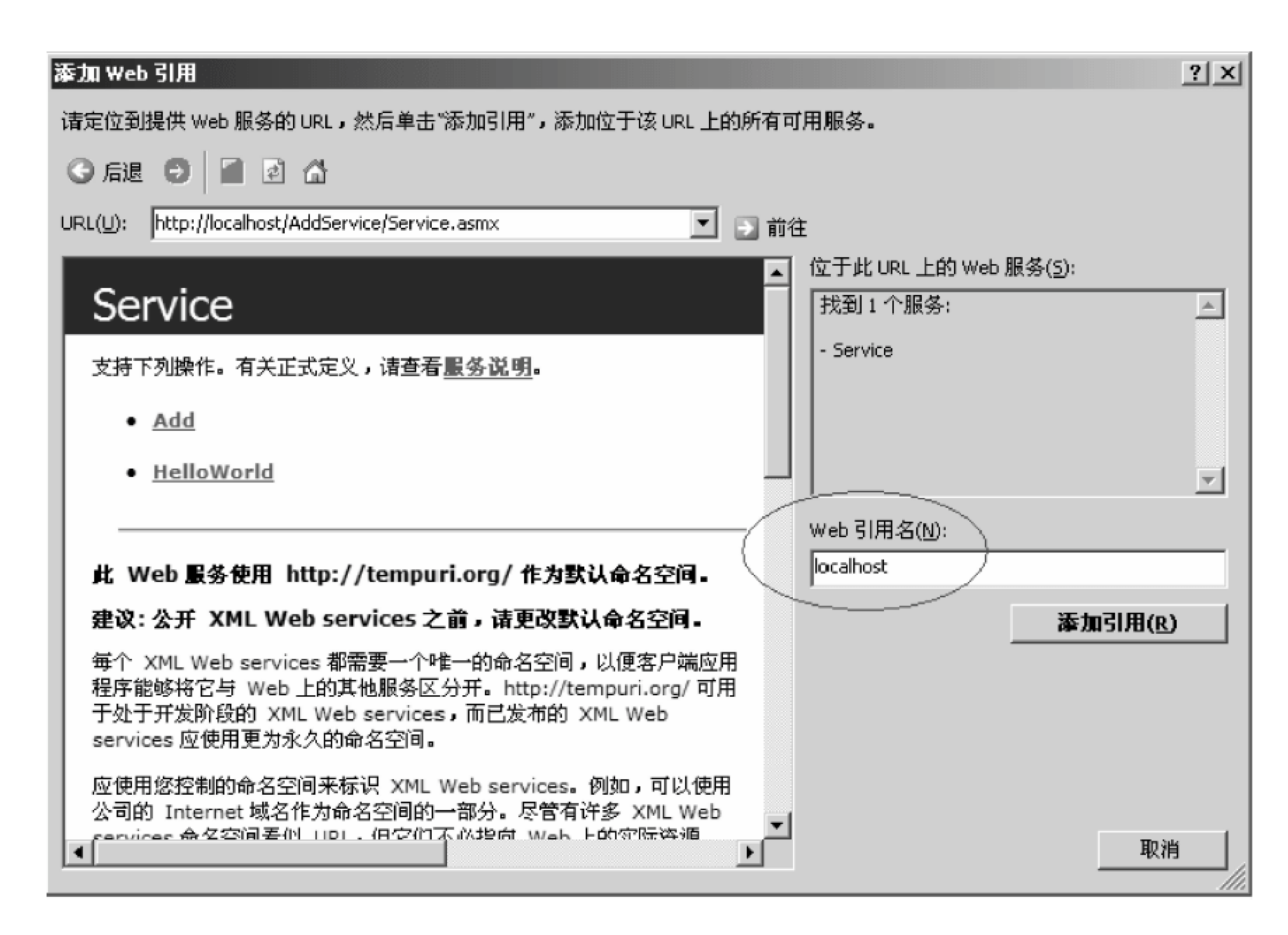


图 11-11 "添加 Web 引用"对话框(2)



图 11-12 添加了 Web 引用 localhost 的网站

```
        //创建 Web Service 代理对象
        localhost.Service s = new localhost.Service();
        //获取用户输入的两个参数
        int x = Convert.ToInt32(txtX.Text);
        int y = Convert.ToInt32(txtY.Text);
        //调用 Web Service 完成两个数的相加
        lblResult.Text = s.Add(x, y).ToString();
    }
```

上述代码中，用方框标出的就是访问 Web Service 的代码。可以看出，这同访问一个本地对象几乎没有什么差别。实际上，在网页中创建的 Web Service 对象是一个代理对象，该代理对象拥有 Web Service 所定义的全部 Web 方法。网页对该代理对象的调用都被传送给远程服务器，服务器处理完之后，再将结果传回调用的网页。

11.5 代 理 类

代理是处于应用程序和 Web Service 之间的一个程序，它负责为应用程序和 Web Service 之间搭建一个桥梁，保证应用程序和 Web Service 之间的通信顺畅。代理类(.dll 文件)通常由 WSDL 语言创建，不包含任何应用程序逻辑，只包含一个传输逻辑——告知用户如何传递参数和检索结果，以及 Web Service 中的方法和原型列表。

客户端应用程序如果使用 Web Service，必须先创建一个代理。代理是要调用的真正代码的替身，负责在计算机边界引导调用。当代理在客户端应用程序中注册后，客户端应用程序调用方法就如调用本地对象一样。代理接受该调用，并以 XML 格式封装调用，然后以 SOAP 请求发送调用到服务器。当服务器返回 SOAP 包给客户端后，代理会对包进行解密，并且如同从本地对象的方法返回数据一样将其返回给客户端应用程序，如图 11-13 所示。

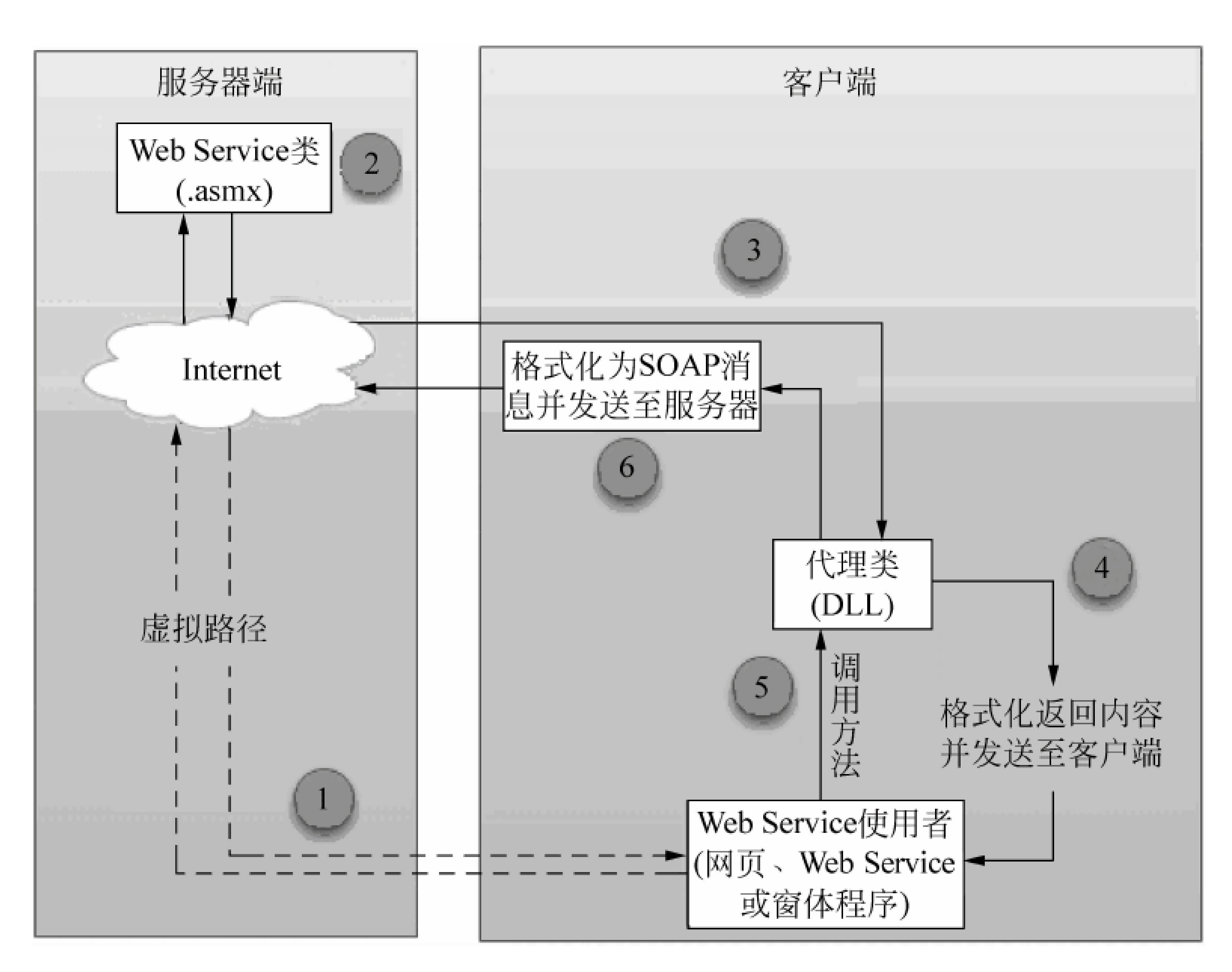


图 11-13　代理的工作过程

11.6 习题与上机练习

1. 选择题

(1) 以下关于发布与部署 Web Service 的说法正确的是(　　)。

A. 发布与部署没有什么区别，两个仅是不同的定义

B. 发布是将 Web Service 放到 IIS 上，部署是制作安装包

C. 发布是将 Web Service 向外界公示，部署是将 Web Service 放到 IIS 上

D. 发布是将 Web Service 的相关信息列入 UDDI 目录中方便查询，而部署仅仅实现了 Web Service 的物理可访问

(2) 在使用 Web Service 之前,常用的 Web Service 发现工具是(　　)。

A. WSDL. exe　　B. Disco. exe

C. Ftp. exe　　D. Ping. exe

(3) XML Web Service 的交互通常使用标准的 Internet 协议不包括(　　)。

A. TCP/IP　　B. HTTP

C. SOAP　　D. IPX/SPX

(4) 以下关于 UDDI 的指定(　　)是错误的。

A. 使用 Web Service 必须通过 UDDI

B. UDDI 能让你的 Web Service 获得更多的使用

C. UDDI 能提供一系列 Web Service 的最终访问点

D. UDDI 负责提供 WSDL 文件

(5) 以下关于代理类和 WSDL 的描述(　　)是错误的。

A. 代理类是对 WSDL 返回内容进行的进一步封装

B. 代理类可通过 WSDL. EXE 自动生成

C. 代理类可替代 WSDL 直接与 Web Service 打交道

D. 代理类中提供了同步和异步调用 Web Service 的方法

(6) 下列关于 Web Services 的描述,(　　)是错误的。

A. Web Services 架构中有三个角色:服务请求者、服务提供者和服务注册中心

B. 服务提供者向服务注册中心发布服务的信息

C. 服务请求者需要向服务注册中心查询其需要的服务的信息

D. 服务请求者需要与服务注册中心绑定以消费服务

(7) Web Services 使用基于(　　)的标准和传输协议交换数据。

A. XSLT　　B. XML　　C. TCP/IP　　D. Java

(8) 下列(　　)是描述 Web 服务的标准 XML 格式。

A. WSDL　　B. UDDI　　C. SOAP　　D. LDAP

(9) Web Services 应用程序具备(　　)特征。

A. 封装性　　B. 松散耦合

C. 使用标准协议规范　　D. 高度可集成性

(10) Web Services 应用的优势体现在(　　)场景中。

A. 跨防火墙通信　　B. 应用程序集成

C. B2B 集成　　D. 数据重用

(11) Web Services 体系结构基于(　　)种逻辑角色 。

A. 服务提供者　　B. 服务注册中心

C. 服务请求者　　D. 消息

2. 简答题

(1) 什么是 Web Service? 它的主要核心技术有哪些?

(2) UDDI 商业注册中心所提供的信息从概念上分为三个部分:白页、黄页、绿页,请说

明它们各代表什么意思?

3. 上机练习

设计并实现一个天气预报查询页。要求调用天气预报 Web Service,其中 Web Service 的地址为 http://www.webxml.com.cn/WebServices/WeatherWebService.asmx。可以选择全国主要城市,并显示该城市三天内的天气情况。

第 12 章 AJAX 技术基础

AJAX 即异步 JavaScript 与 XML 技术(Asynchronous JavaScript and XML,AJAX),是 Html、JavaScript、DHTML、DOM、XML 等技术的组合体,这一组合改变了以往 Web 界面的交互方式,带来了更加良好的用户体验,已成为 Web 2.0 时代广泛应用的一项技术。

12.1 AJAX 概述

目前的应用程序主要有两种工作模式:桌面应用和 Web 应用。前者被安装在本地计算机上运行;后者被安装在服务器上,通过 Web 浏览器进行访问。桌面应用一般具有极快的运行速度和良好的交互能力,而 Web 应用往往达不到同样的性能,这主要是由于传统 Web 技术中,基于"请求-刷新"的工作机制存在着以下一些固有缺陷:

- 页面需要反复刷新:每当用户向服务器端请求一次数据时,都会导致页面整体刷新,哪怕只是更新一个数据,服务器端也会重新生成整个页面,造成很大的服务器负荷及传输数据量。很多时候,这种消耗是没有意义的。
- 页面刷新期间,用户只能等待:浏览器和服务器采用同步的通信机制,当用户提交请求后,页面将处于冻结状态,等待服务器端反馈数据。在新页面加载之前,用户只能等待,不能做任何事情;尤其是网速较慢时,会消耗很长时间。
- 绝大多数工作都在服务器端完成,服务器负荷很重,往往成为系统性能的瓶颈。

AJAX 技术的出现改变了这种状况,具有如下特点:

(1) 可实现页面的局部刷新:可以借助客户端技术,找到页面中需要更新的局部区域,只更新该区域中的数据,而不需要刷新整个页面,大大减轻服务器的负荷。

(2) 只做必要的数据交换:由于只刷新页面的部分区域,就不需要服务器端生成整个 HTML 页面,而是只生成客户端需要的部分数据,大大降低网络传输的数据量。

(3) 异步访问服务器端:当用户提交请求时,浏览器和服务器采用异步通信的方式在后台处理请求,页面不会冻结;在请求结果返回前,用户可以继续浏览网页而不用傻等;当服务器端返回结果后,客户端再自动刷新页面。

AJAX 编程分为服务器端与客户端两部分,如图 12-1 所示。

服务器端编程可以使用现有的技术(如 ASP.NET),但由于不用返回整个页面,只返回需要的数据,所以很多时候使用 Web Service 或 Http Handler 处理 AJAX 请求,按指定格式将数据打包发回客户端即可。

客户端编程的核心是 XmlHttpRequest 对象,由它向服务器端发送异步请求,并接收返回的数据;然后使用文档对象模型在文档中检索局部更新的区域,并用服务器端返回的数

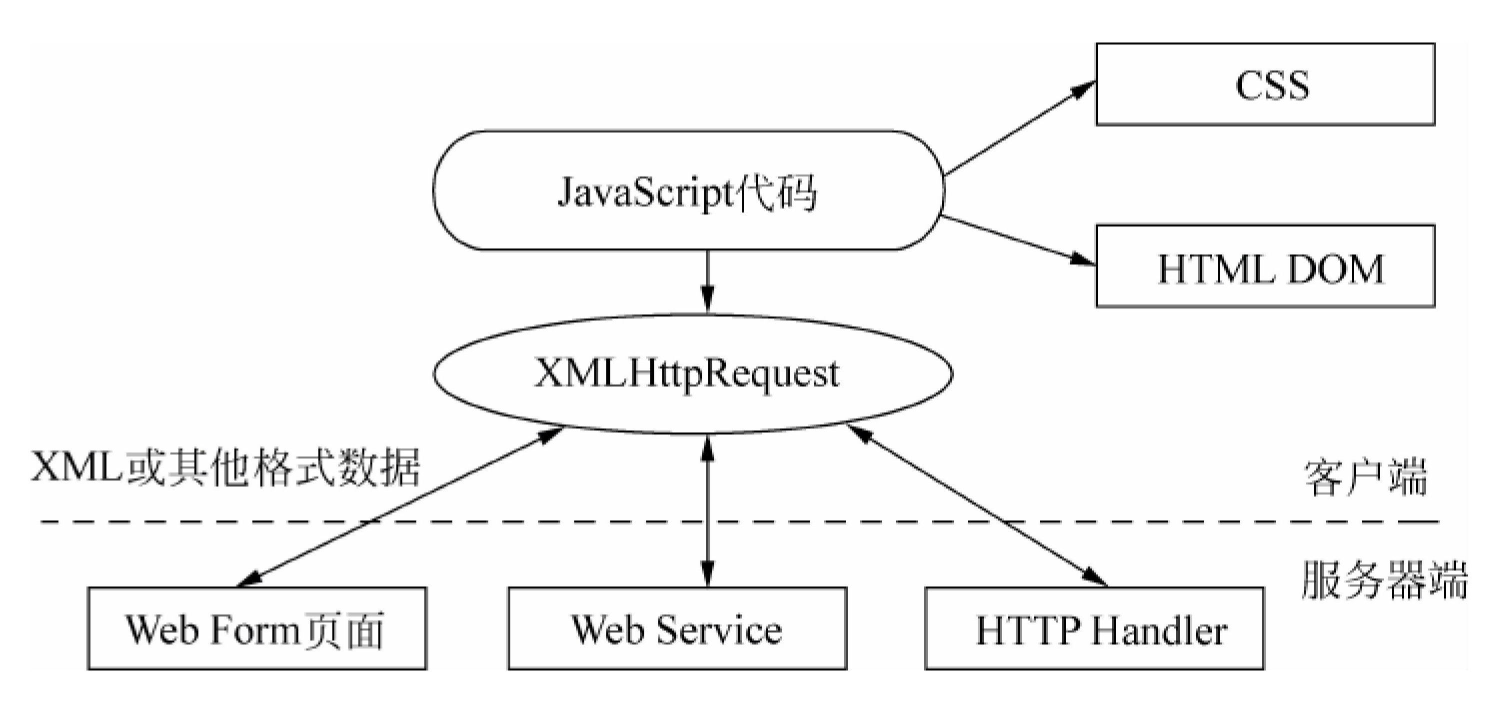


图 12-1　AJAX 技术框架

据更新页面。

在 ASP.NET 下开发 AJAX 风格的应用程序可以采用两种方法：一是手工编码方式，即手工完成异步通信及局部刷新的处理过程；二是使用 AJAX 控件的方式。

12.2　手工编码的 AJAX 应用

传统的 AJAX 应用需要在客户端手工编码，实现异步通信及局部刷新的处理过程。

例 12-1　使用 AJAX 技术从服务器端获取 XML 内容并显示在页面上。

(1) 创建网站项目，建立如下的 Employees.XML 文件。

```
<table border = "1">
    <tr><th>EmpID</th><th>FirstName</th><th>LastName</th></tr>
    <tr><td>1</td><td>Tom</td><td>Cat</td></tr>
    <tr><td>2</td><td>Jerry</td><td>Mouse</td></tr>
</table>
```

(2) 建立测试页面 ShowEmployees.aspx，代码如下：

```
<html xmlns = "http://www.w3.org/1999/xhtml">
<head><title>使用 AJAX 显示 XML 数据</title></head>
<body>
    <form action = "#">
    <input type = "button" value = "获取员工信息并显示" onclick = "startRequest();" />
    <div id = "results" ></div>
    </form>
</body>
</html>
```

该页面非常简单，只有一个按钮和一个 DIV 元素。单击按钮时向服务器发送异步请求，获取 Employees.XML 文件中的内容，并将其显示在 DIV 元素中。

(3) 定义函数以创建 XMLHttpRequest 对象。在页面的 head 标记中加入如下代码：

```
<script type = "text/javascript">
var xhr;
```

```
function CreateXMLHttpRequest()
{
    try
    {
        xhr = new XMLHttpRequest();
    }catch(err)
    {
        xhr = new ActiveXObject("Microsoft.XMLHTTP");
    }
}
</script ">
```

XMLHttpRequest 对象是 AJAX 技术的基石,首先需要安全的创建该对象。尽管现代的浏览器对 XMLHttpRequest 对象有广泛的支持,但在如何创建和访问这个对象上还是有着细微的差别。在 FireFox 及 IE 7.0 以上浏览器上,该对象都是作为本地 JavaScript 对象实现的,但在 IE 7.0 以前的版本中,却是作为 ActiveX 对象实现的。因为这些差别,使用 JavaScript 创建 XMLHttpRequest 对象实例时必须要有足够的智能,采用正确的方法。本例中先尝试按照本地对象来创建,若抛出异常,则说明浏览器版本较老,所以再按照 ActiveX 方式创建,这样无论如何保证正确创建了一个 XMLHttpRequest 对象实例。

(4) 在按钮单击事件中触发异步请求。在 script 标记中加入以下脚本:

```
function startRequest()
{
      CreateXMLHttpRequest();
      xhr.open("GET", "EmployeesData.xml");
      xhr.onreadystatechange = showdata;
      xhr.send(null);
}
```

在函数体中,第 1 行创建了 XMLHttpRequest 对象实例。

第 2 行调用 XMLHttpRequest 对象的 open 方法建立异步调用,即定义要发送到服务器的请求。这里需要两个参数:请求方式(GET、POST 或 PUT)及请求的 URL。还可以使用第 3 个参数指明请求是否异步执行,默认为 True,所以省略。

第 3 行指明如何处理响应,即当异步请求执行完成,返回请求数据后,由 showdata 函数负责刷新页面。

第 4 行真正的发送了异步请求。Send 方法可以接收一个字符型参数,代表随请求发送的额外数据。对于 IE 可以忽略该参数,但对于 FireFox,必须提供一个 null 引用,否则回调处理可能会不正确。

(5) 处理响应。

在 script 标记中加入以下脚本:

```
function showdata()
{
      if (xhr.readyState == 4)
      {
```

```
                if (xhr.status == 200)
                {
                    document.getElementById("results").innerHTML = xhr.responseText;
                }
            }
        }
```

当响应被返回时，可以使用 responseText 和 responseXML 属性从 XMLHttpRequest 对象中析取需要的数据。从字面意思理解，responseText 将数据内容当做一个字符串返回，而 responseXML 则将其作为树的节点对象返回，具体应用中要根据实际情况来进行选择。

特别需要说明的是，在获取响应数据之前，必须先判断 XMLHttpRequest 对象的状态，确认请求已处理完成并且正确返回了响应数据。在步骤(3)中注册了当 XMLHttpRequest 对象的 readyState 状态改变时将触发 showdata 函数调用，但 readState 共有 5 种状态，任何一次状态变化都会触发 showdata，而只有状态值为 4 时才表示响应完全加载。当响应完全加载时，还要判断本次请求处理是否成功，通过 status 属性判断，值为 200 表示处理成功，其他代码都表示出现了某种类型的错误。

通过本例可以看出，AJAX 是一项比较中立的技术，不在乎服务器端采用什么技术，只要能够返回 XML 格式的数据即可，本例更是没有使用任何服务器端编程技术，直接从一个文件中获取 XML。事实上，AJAX 在出现的早期要求服务器端返回 XML 格式的数据，由于 XML 数据往往格式冗长，现在的应用中往往也不使用 XML 返回结果。

例 12-2 使用 AJAX 技术实现列表框的联动。

在第 1 个列表框中显示大区(Region)列表，第 2 个列表框中显示地区(Territory)列表，当在第 1 个列表框中选择某个大区时，第 2 个列表框中自动填充该大区下的所有地区。页面的运行效果如图 12-2 所示。

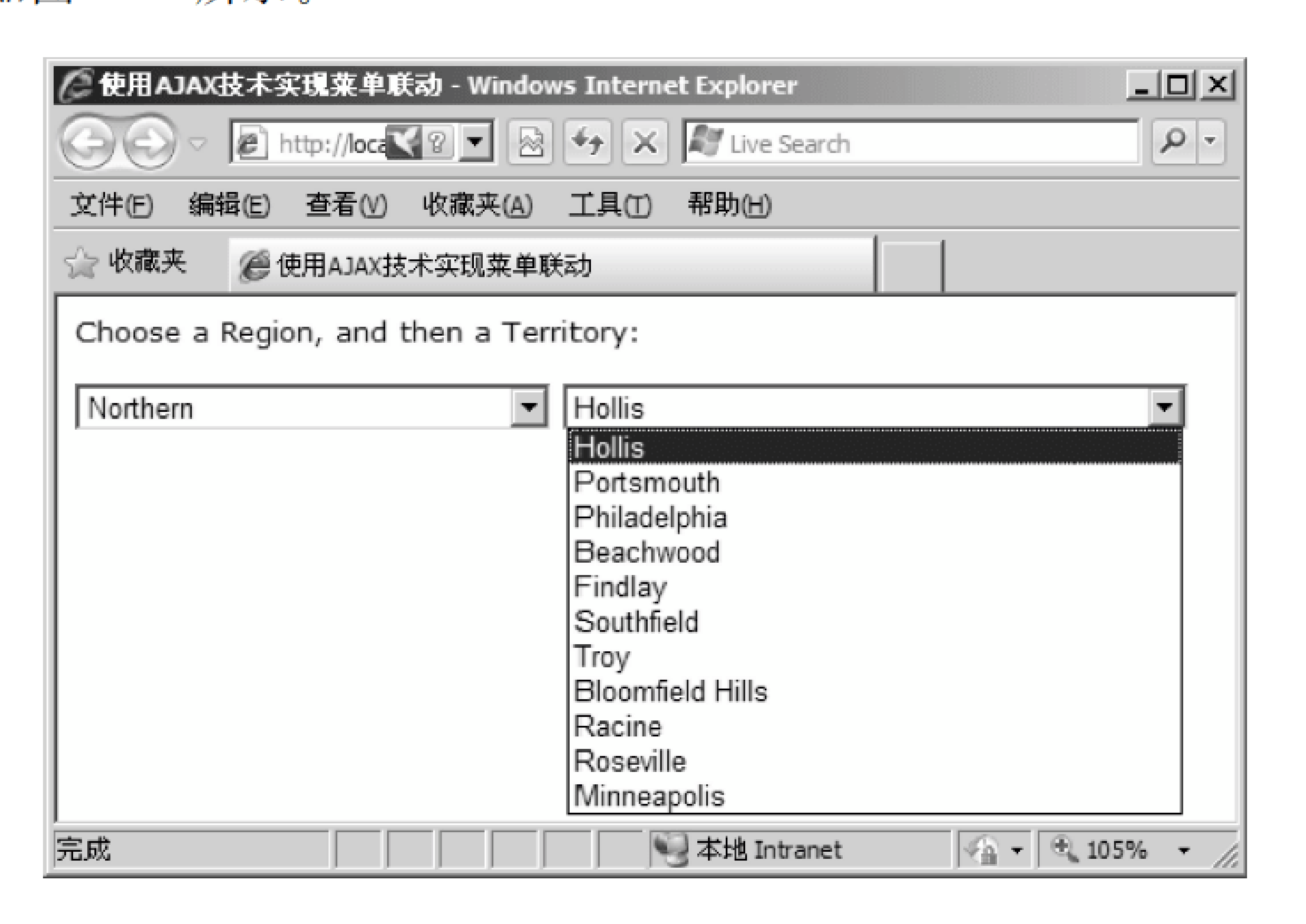


图 12-2 使用 AJAX 技术实现列表框联动

用常规方式实现列表框联动也比较简单，只要设置大区列表框的 AutoPostBack 属性为真，然后在服务器端捕获其 SelectedIndexChanged 事件，在其中编码，访问数据库并填充

地区列表框即可。这种方式会导致页面整体刷新，产生不必要的延迟和闪烁，使用 AJAX 技术实现效果更佳。

(1) 建立页面，代码如下：

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>使用 AJAX 技术实现列表框联动</title>
    <script type="text/javascript">
        var xmlRequest;
        function CreateXMLHttpRequest()
        {
          try
          {
            xmlRequest = new XMLHttpRequest();
          }
          catch(err)
          {
            xmlRequest = new ActiveXObject("Microsoft.XMLHTTP");
          }
        }
    </script>
</head>
<body onload="CreateXMLHttpRequest();">
<form id="form1" runat="server">
     Choose a Region, and then a Territory:<br /><br />
     <asp:DropDownList ID="lstRegions" runat="server" DataSourceID="sourceRegions"
          DataTextField="RegionDescription" DataValueField="RegionID"
          onchange="getTerritories();"></asp:DropDownList>
     <asp:DropDownList ID="lstTerritories" runat="server" />
     <asp:SqlDataSource ID="sourceRegions" runat="server"
          ConnectionString="<%$ ConnectionStrings:Northwind %>"
          SelectCommand="SELECT RegionID, RegionDescription FROM Region">
     </asp:SqlDataSource>
</form>
</body>
</html>
```

本例在服务器端使用 SqlDataSource 自动填充大区下拉列表框。客户端在页面加载完成后自动创建 XMLHttpRequest 对象，并捕获大区下拉框的 change 事件以触发 AJAX 调用。

(2) 触发 AJAX 通信，函数代码如下：

```
function getTerritories ()
{
      var val = document.getElementById('lstRegions').value;
      var url = "ajaxddlhandler.ashx?regid=" + val;
      xmlRequest.open("GET", url);
      xmlRequest.onreadystatechange = RefreshDDL2;
      xmlRequest.send(null);
}
```

函数第一行在 DOM 中查找大区下拉列表框，并获取其当前选项值。下一步是向服务器发送异步请求，获取该大区下的所有地区列表。由于服务器端不需生成整个页面，没有必要采用复杂的 Web Form 模型，这里使用了一般 HTTP 处理程序来处理异步请求，所以请求的 URL 为 ajaxddlhandler.ashx，同时将大区代码作为参数传递过去。

(3) 创建一般 HTTP 处理程序以处理异步请求。

在 ASP.NET 中，所有的请求最终都是由一个实现了 IHttpHandler 接口的类来处理的，一般情况下，这个类就是一个 Web Form 页面。

但很多时候，只需要接收和处理请求，然后返回数据，不必使用基于控件的 Web Form 模型，不需要走完整的页面事件过程来创建页面(包括创建网页对象、持久化视图状态等)，这样可以节约大量的服务器资源。这时，使用低层接口会非常方便，可以自定义一个 Http 处理程序，实现 IHttpHandler 接口，这样就可以访问 Request 和 Response 对象，实现简单的请求处理。

IHttpHandler 接口中定义了两个成员，如表 12-1 所示。

表 12-1 IHttpHandler 接口的成员

成　员	描　述
ProcessRequest	请求处理方法，当请求该处理程序时会自动调用该方法。在该方法中，可以通过传入的 HttpContext 对象访问 Request、Response 等内部对象
IsReusable	该属性指明本处理程序是否可以重用。ProcessRequest 方法执行完成后会自动检查该属性，若为假，则立刻释放该对象，若为真则不释放，还可以被另一个和当前请求类型相同的请求所重用

自定义的请求处理程序代码如下：

```
<%@ WebHandler Language="C#" Class="AjaxDDLHandler" %>
using System;
using System.Web;
using System.Text;
using System.Data.SqlClient;
using System.Web.Configuration;
public class AjaxDDLHandler : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        HttpResponse response = context.Response;
        context.Response.ContentType = "text/plain";
        string regid = context.Request.QueryString["regid"];
        string territories = GetTerritories (regid);
        context.Response.Write(territories);
    }
    public bool IsReusable
    {
      get{return true; }
    }
    public string GetTerritories(string regid)
```

```
        {
            SqlConnection con = New SqlConnection(
            WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString);
            SqlCommand cmd = new SqlCommand(
                "SELECT * FROM Territories WHERE RegionID = @RegionID", con);
            cmd.Parameters.AddWithValue("@RegionID", regid);
            StringBuilder results = new StringBuilder();
            try
            {
                  con.Open();
                  SqlDataReader reader = cmd.ExecuteReader();
                  while (reader.Read())
                  {
                      results.Append(reader["TerritoryDescription"]);
                      results.Append("|");
                      results.Append(reader["TerritoryID"]);
                      results.Append("||");
                  }
            reader.Close();
            }
            catch (SqlException err){ … }
            finally { con.Close(); }
            return results.ToString();
        }
    }
```

在 ProcessRequest 方法中，设置相应的内容类型为 text/plain，表示以纯文本的形式返回数据；然后从请求参数中获取 RegionID，并调用 GetTerritories 方法以检索该大区下所有地区的信息，最后将查询结果写到响应中，返回个客户端。

在 GetTerritories 方法中，根据一个 RegionID 可以从数据库中检索到一系列 Territory 信息，按传统的做法，可以将这些信息包装在一段 XML 中返回，但考虑到 XML 需要复杂的标记，会增加传输的数据量，所以这里使用了更简单的数据组织方式。使用 StringBuilder 构造返回字符串，各条 Territory 之间用“||”符号分隔，每条 Territory 中含 TerritoryDescription 和 TerritoryID 两个数据项，之间用“|”符号分隔。

（4）处理返回结果，刷新页面，代码如下：

```
    function RefreshDDL2()
    {
        if (xmlRequest.readyState == 4)
        {
            if (xmlRequest.status == 200)
            {
                 var result = xmlRequest.responseText;
                 var lstTerritories = document.getElementById("lstTerritories");
                 lstTerritories.innerHTML = "";
                 var rows = result.split("||");
                 for (var i = 0; i < rows.length - 1; ++i)
```

```
                {
                    var fields = rows[i].split("|");
                    var territoryDesc = fields[0];
                    var territoryID = fields[1];
                    var option = document.createElement("option");
                    option.value = territoryID;
                    option.innerHTML = territoryDesc;
                    lstTerritories.appendChild(option);
                }
            }
        }
    }
```

这里首先从 XMLHttpRequest 对象中获得返回的数据(纯文本字符串),然后要对该字符串进行解析。在 JavaScript 中调用 string 类型的 split 方法可以实现字符串的切分,传入的参数为字符串分隔符。首先按"||"分隔,可以将结果拆分成一系列 Territories;每个 Territory 又包含两个数据项,所以再使用"|"切分,得到 TerritoryDescription 和 TerritoryID,在循环体中,以这些数据为基础生成选项,添加到第二个下拉列表框中,至此,AJAX 请求处理完成。

运行程序,在第一个列表框中选择一个大区,可以看到页面并没有闪动,但第二个列表框中已经填入了适当的列表,这就是异步通信带来的用户体验。

12.3 使用 Ajax Extensions 快速构建 Ajax 应用

前面使用传统编程方式构建 AJAX 应用,客户端需要编写大量的 JavaScript 代码,还需要弥补 ASP.NET 服务器端抽象和客户端 HTML DOM 之间的鸿沟,开发难度较大。

为简化开发,可以使用 Microsoft Ajax 技术。它提供了一套服务器端编程模型,能够自动产生各种需要的客户端代码(即使用 XMLHttpRequest 对象执行异步请求的代码),这样就可以非常方便地创建高度交互 AJAX 风格的页面了。

12.3.1 Microsoft Ajax 概述

Microsoft Ajax 技术体系分为两大模块:客户端模块和服务器端模块,如图 12-3 所示。

在客户端可使用 Microsoft Ajax Library,这是一组独立的 JavaScript 库文件,提供了易用的客户端组件,并提供浏览器兼容性支持、异步通信服务及调试和错误处理等基础服务。由于该库独立于服务器端技术,可以单独使用客户端脚本而不使用 ASP.NET 服务器端控件,或使用其他的服务器端技术。

服务器端模块主要包含一组服务器端控件,另外还提供了应用程序服务(如角色和认证信息服务等)、Web Service 支持(允许在客户端脚本中调用 Web Service)及脚本支持等功能。使用 Ajax 服务器端模块,可以像以往开发服务器端程序一样将 Ajax 控件添加到网页上,该页会自动将支持的客户端脚本发送到浏览器以获取 Ajax 功能。

在服务器端模块,系统本身提供了 4 个最常用的 Ajax 控件,如表 12-2 所示。

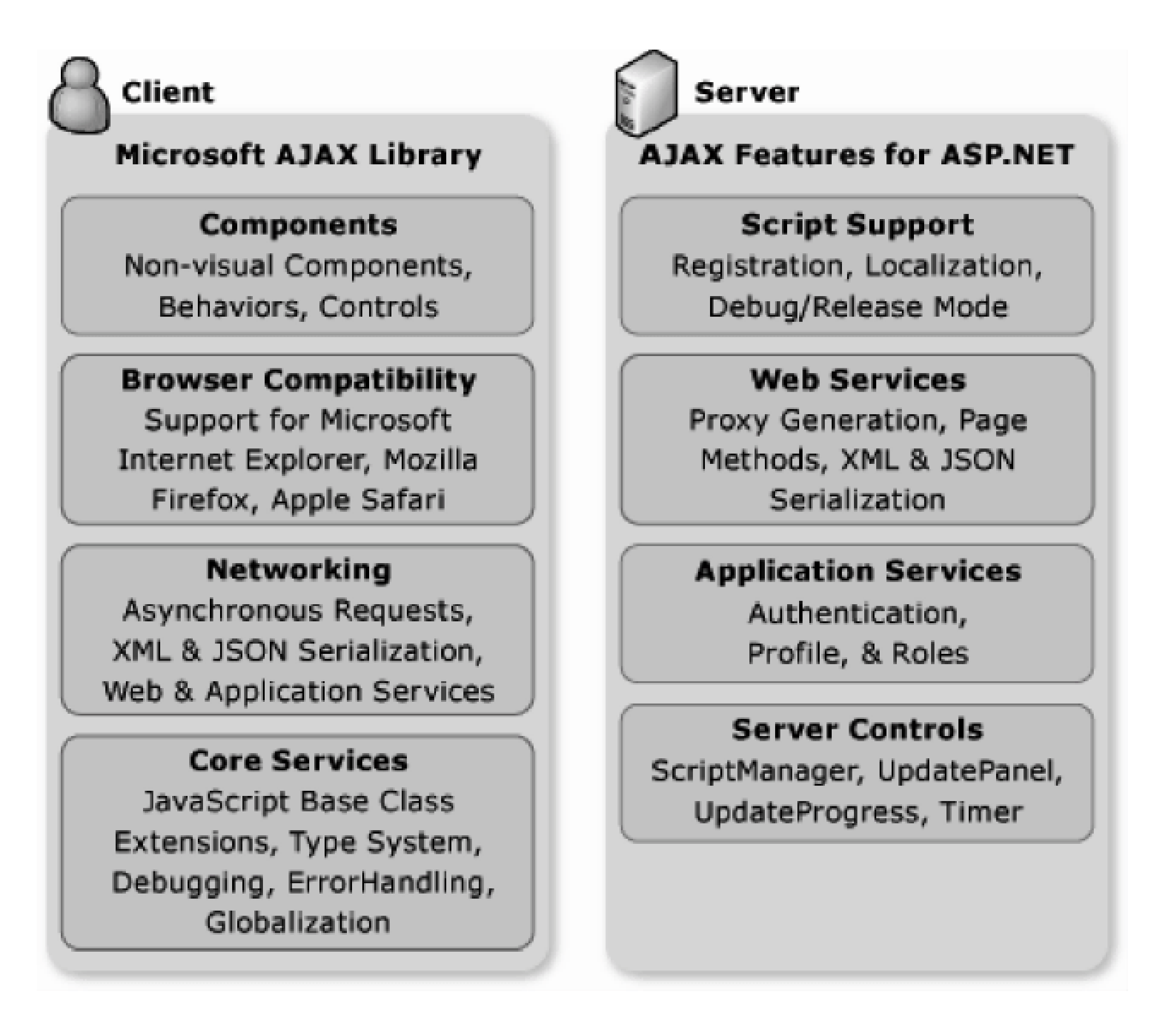


图 12-3　Microsoft Ajax 体系结构

表 12-2　常用的 Ajax 控件

控　　件	描　　述
ScriptManager	管理客户端组件、部分页呈现、本地化、全球化和自定义用户脚本的脚本资源。只要使用 UpdatePanel、UpdateProgress 和 Timer 控件，就一定要创建 ScriptManager 来管理脚本。若只使用客户端模块，则不需要创建该对象
UpdatePanel	最重要的服务器端控件，能够刷新页面的选定部分，而不是使用同步回发来刷新整个页面
UpdateProgress	提供有关 UpdatePanel 控件中的部分页更新的进度状态信息
Timer	按定义的时间间隔执行回发。可以使用该控件来发送整个页，或将其与 UpdatePanel 控件一起使用以按定义的时间间隔执行部分页更新

12.3.2　使用 UpdatePanel 控件实现页面局部刷新

异步请求和局部刷新是 AJAX 应用的两个基本特征，若要实现这两项功能，通常需要深入研究 JavaScript，仔细处理客户端与服务器端的交互。不过，若使用 UpdatePanel 控件，不用编写任何 JavaScript 代码，不用考虑复杂的通信过程，就能开发出功能强大的 AJAX 应用，大量的客户端代码及底层通信服务都由服务器端控件自动完成。

例 12-3　使用 UpdatePanel 实现页面的局部更新。

(1) 创建 PartialUpdate.aspx 页面，在控件工具箱中单击 AJAX Extensions 组，从中选择 ScriptManager 控件，双击将其加入页面中。

(2) 从控件工具箱中选择 AJAX Extensions 组中的 UpdatePanel 控件，双击加入页面中。

(3) 单击 UpdatePanel 控件内部，然后在工具箱的“标准”选项卡中双击 Label 和 Button 控件以将它们添加到 UpdatePanel 控件中。

(4) 在 UpdatePanel 外再添加一个 Label 和一个 Button 控件。

(5) 调整各控件的属性,最终生成的页面代码如下:

```
<form id = "form1" runat = "server">
<asp:ScriptManager ID = "ScriptManager1" runat = "server"></asp:ScriptManager>
<h2>页面局部更新示例</h2>
<hr />
<asp:UpdatePanel ID = "UpdatePanel1" runat = "server" UpdateMode = "Conditional">
    <ContentTemplate>
        局部更新的时间:
        <asp:Label ID = "Label1" runat = "server" Font - Bold = "True"></asp:Label>
        <asp:Button ID = "Button1" runat = "server" Text = "更新本区域的时间" />
    </ContentTemplate>
</asp:UpdatePanel>
<hr />
整体更新的时间:
<asp:Label ID = "Label2" runat = "server" Font - Bold = "True"></asp:Label>
<asp:Button ID = "Button2" runat = "server" Text = "更新整个页面的时间" />
</form>
```

(6) 在后台代码文件中加入如下的 Page_Load 事件过程:

```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = DateTime.Now.ToLongTimeString();
    Label2.Text = DateTime.Now.ToLongTimeString();
}
```

运行该程序,界面如图 12-4 所示。



图 12-4 页面局部刷新示例

单击下排的按钮,能够看到页面被提交并整体刷新,两个 Label 上显示的时间都发生了变化。但单击第一个按钮,会发现页面没有闪烁(没有整体刷新),且只有第一个 Label 上显示的时间发生了变化,而第二个 Label 没有变化。这就可以说明,放置在 UpdatePanel 中的控件,会自动产生异步请求,并实现局部刷新,这正是 AJAX 技术的核心特性,但是在本示例中,没有写一行的 JavaScript 代码,也没有操作 XMLHttpRequest 对象,一切都是由

AJAX 框架自动完成的。ASP.NET AJAX Extensions 帮助程序员完成了最困难、最复杂的底层工作，大大简化了 AJAX 开发过程。

实现页面局部刷新与异步请求的核心是 ScriptManager 和 UpdatePanel 控件。前者用于控制服务器端与客户端的交互，解决 JavaScript 脚本下载以及客户端与服务器之间的通信问题，每个使用 AJAX Extensions 的网页都需要一个 ScriptManager 控件；后者用来定义页面上的可更新区域，当一次异步回发发生时，该区域会得到更新。在一个页面上可以放置多个 UpdatePanel 控件以定义多个可更新区域，甚至可以嵌套的使用 UpdatePanel 控件。

UpdatePanel 控件使用模板的方式定义其显示内容。它有一个 ContentTemplate 模板，可以容纳各种页面元素。对于那些可以引发 PostBack 的传统 ASP.NET 控件（如 Button 控件），只要被放置在 UpdatePanel 中，不用作任何设置，这些标准回发就会自动转换为异步回发，这就是 UpdatePanel 的神奇之处。

需要特别说明的是，当异步回发发生时，服务器端的处理过程与传统回发的处理过程没有任何区别，仍然会经历传统回发一样的生命周期，所以使用异步回发并没有减少服务器端的运算负荷。在 Page_Load 事件过程中，从代码看是要更新两个标签上的时间，但实际运行后只更新了 UpdatePanel 范围内的时间，可见服务器在给浏览器回传数据时是有区别的。在异步模式下，服务器会根据请求，仅发送要更新区域（即 UpdatePanel 上）的数据；而在同步回发时，服务器会发回整个页面。

当一个页面上有多个 UpdatePanel 时，一个 Panel 上的异步回发是否会影响另一个 Panel 上的刷新由 UpdatePanel 控件的 UpdateMode 属性决定，该属性有两种选择：当选择 Always 时，任何一个回发（包括同步回发和异步回发）都会更新其内容；当选择 Conditional 时，仅满足条件的回发更新其内容。该属性默认值为 Always。

例 12-4 包含多个 UpdatePanel 的页面局部更新示例。

建立如下的页面：

```
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
<h2>局部更新方式示例</h2><hr />
整页回发的时间：
<asp:Label ID="Label1" runat="server" Font-Bold="True"></asp:Label>
<asp:Button ID="Button1" runat="server" Text="传统提交" /><hr />
<asp:UpdatePanel ID="PanelAlways" runat="server">
    <ContentTemplate>
        异步回发 Always 模式：
        <asp:Label ID="Label2" runat="server" Font-Bold="True"></asp:Label>
        <asp:Button ID="Button2" runat="server" Text="异步提交" />
    </ContentTemplate>
</asp:UpdatePanel><hr />
<asp:UpdatePanel ID="PanelConditional" runat="server" UpdateMode="Conditional">
    <ContentTemplate>
        异步回发 Conditional 模式：
        <asp:Label ID="Label3" runat="server" Font-Bold="True"></asp:Label>
        <asp:Button ID="Button3" runat="server" Text="异步提交" />
    </ContentTemplate>
```

```
</asp:UpdatePanel>
</form>
```

在后台建立如下的 Page_Load 事件过程：

```
protected void Page_Load(object sender, EventArgs e)
{
      Label1.Text = DateTime.Now.ToLongTimeString();
      Label2.Text = DateTime.Now.ToLongTimeString();
      Label3.Text = DateTime.Now.ToLongTimeString();
}
```

运行该程序，效果如图 12-5 所示。

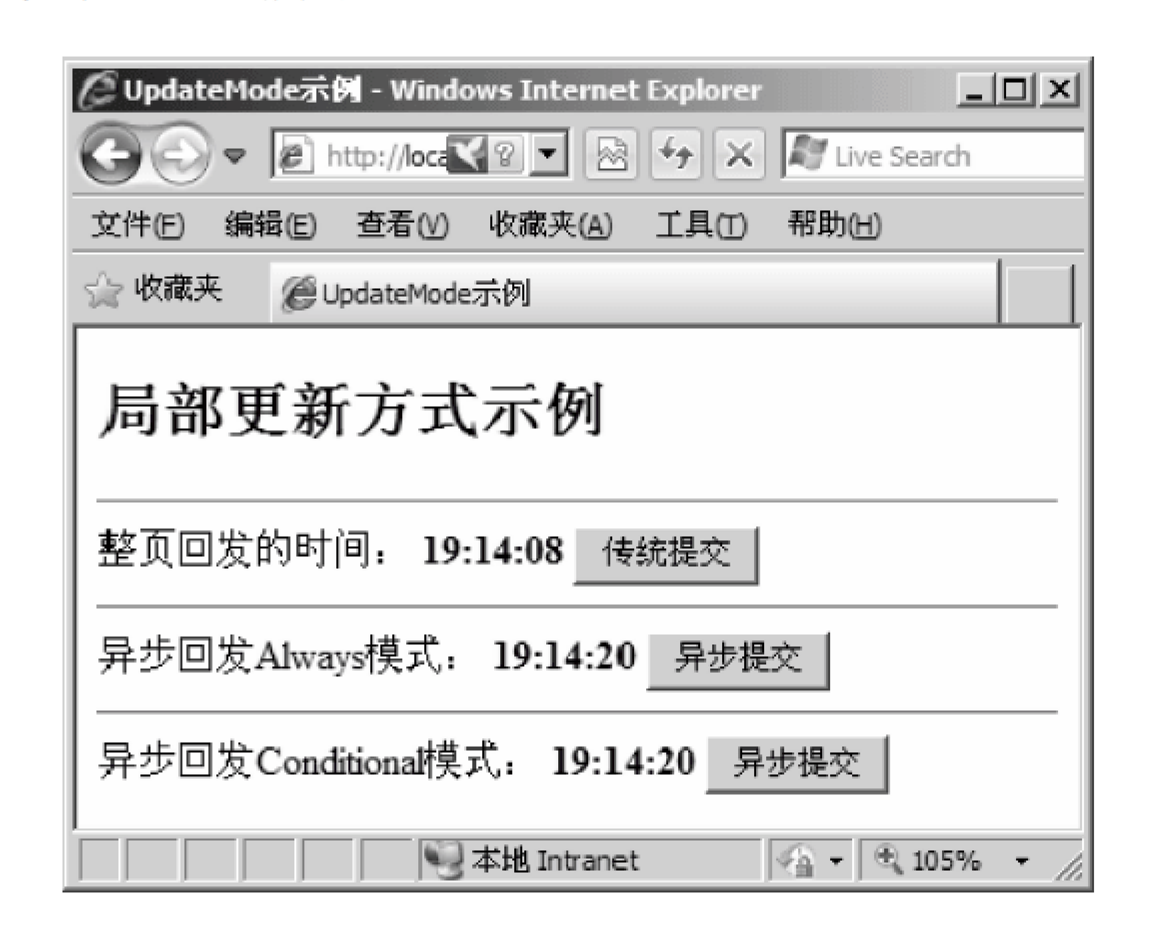


图 12-5 异步回发模式示例

单击第 1 个按钮，会引发同步回发，这时 3 个 Label 上的时间都会刷新。

单击第 2 个按钮，在 PanelAlways 上产生异步回发，只有该 Panel 上的时间被刷新。

单击第 3 个按钮，在 PanelConditional 上产生异步回发，后两个时间被刷新。

可以看到，PanelAlways 面板的 UpdateMode 属性被设置为 Always，这样，任何一次异步回发都会导致该面板的内容刷新；而 PanelConditional 面板的 UpdateMode 属性设置为 Conditional，该面板上的异步回发事件能引发自身的更新，而其他面板的异步回发不能引起它的刷新。

前面的示例都是由 UpdatePanel 内部的控件引起异步回发，最常见的是由按钮控件引发，而 AutoPostBack 属性为真的许多别的控件也能引起异步回发。若不希望 UpdatePanel 内部控件引发的异步回发来更新此 Panel，可以将该 Panel 的 ChildrenAsTriggers 属性设置为 False。

在某些情况下，可能希望由放置在 UpdatePanel 外的某个控件引发异步回发而更新此 Panel，那么该控件被称为异步回发触发器（AsyncPostBackTrigger）。一个 UpdatePanel 可以注册一个或多个异步回发触发器，当一个控件被注册为异步回发触发器时，该控件的回发将不再是标准回发，而是自动被转为异步回发。

使用异步回发触发器的代码声明如下：

```
<asp:UpdatePanel ID = "UpdatePanel1" runat = "server" UpdateMode = "Conditional"
        ChildrenAsTriggers = "false">
    <ContentTemplate>
        局部更新区域的时间：<% = DateTime.Now.ToString() %>
    </ContentTemplate>
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID = "Button1" />
    </Triggers>
</asp:UpdatePanel>
<asp:Button ID = "Button1" runat = "server" Text = "更新时间" />
```

现在可以总结 UpdatePanel 的更新规则了。当 UpdateMode 属性设为 Always 时，页面上任何回发都会导致 UpdatePanel 的内容更新。若其 UpdateMode 属性设置为 Conditional，则以下情况下，该 Panel 的内容才会被更新：

- ChildrenAsTriggers 属性设置为 True，并且任何一个位于该 UpdatePanel 内部的控件引起回发。
- 直接在代码中调用 UpdatePanel 控件的 Update 方法。
- 如果此 UpdatePanel 嵌套在另一个 UpdatePanel 中，那么父 Panel 的更新将导致子 Panel 的更新。
- 任何一个注册为本 UpdatePanel 的触发器控件引发的回发（无论该 UpdatePanel 的 ChildrenAsTriggers 属性是 True 还是 False）。

12.3.3 使用 UpdateProgress 控件显示更新进度

由于 AJAX 页面采用局部刷新，对于一些耗时较长的工作，用户可能不知道页面正在向服务器请求数据，比较人性化的做法是在请求开始时给用户一个提示信息，而在返回数据后这些信息自动消失。使用 UpdateProgress 控件能够实现这样的功能。

从本质上说，UpdateProgress 并不是真正的进度指示，只提供一条等待信息让用户知道页面正在做后台处理，这些等待信息可以是静态的消息或图片，通常会用一个 GIF 动画来模拟进度条。

例 12-5 使用 UpdateProgress 控件显示等待信息。

该页面的运行效果如图 12-6 所示，页面代码如下：

```
<form id = "form1" runat = "server">
<asp:ScriptManager ID = "ScriptManager1" runat = "server"></asp:ScriptManager>
<h2>使用 UpdateProgress 控件显示等待信息</h2><hr />
<asp:UpdatePanel ID = "uppnl1" runat = "server">
    <ContentTemplate>
        <asp:Button ID = "Button1" runat = "server" Text = "启动一个耗时的任务"
            onclick = "Button1_Click" />
        <asp:Label ID = "lblmsg" runat = "server" Font - Bold = "True"></asp:Label>
    </ContentTemplate>
</asp:UpdatePanel><br />
<asp:UpdateProgress runat = "server" id = "uppg1" AssociatedUpdatePanelID = "uppnl1">
  <ProgressTemplate>
```

```
            正在处理< img src = "wait.gif" alt = "Wait" />
    </ProgressTemplate >
</asp:UpdateProgress >
</form >
```

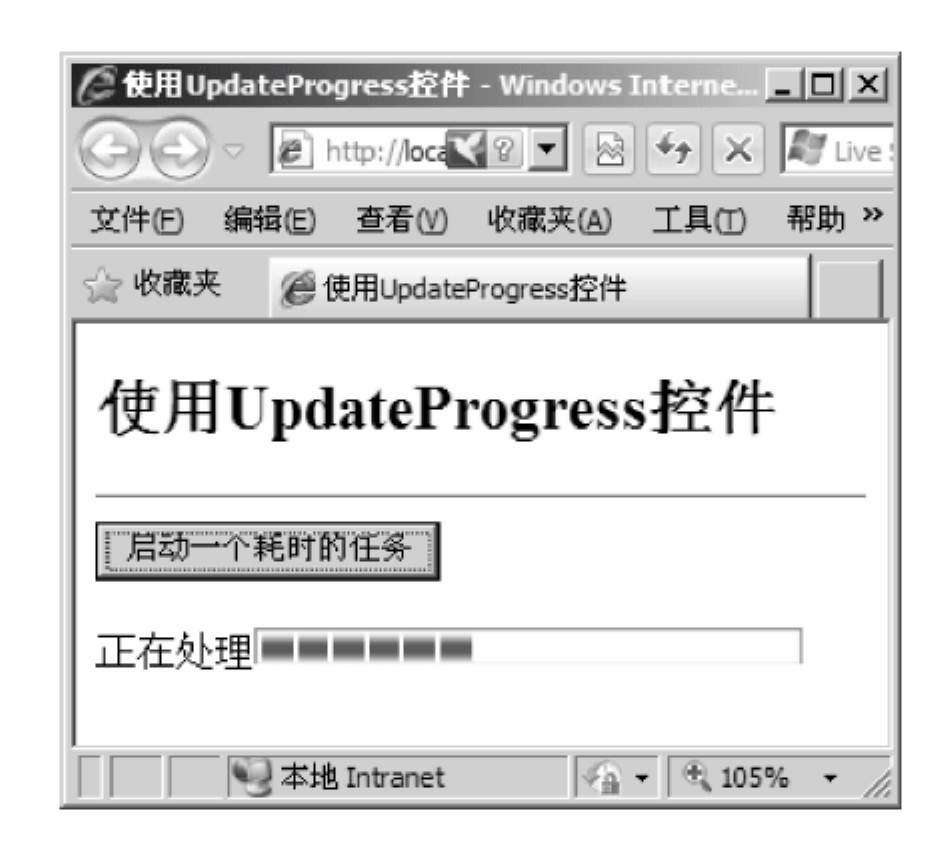


图 12-6 使用 UpdateProgress 控件的页面运行效果

可以看到，UpdateProgress 控件使用模板设计其显示样式，本例在 ProgressTemplate 模板中显示了几个文字及一个动画图标。

本例还将 UpdateProgress 控件的 AssociatedUpdatePanelID 属性关联到 UpdatePanel 控件上，实际上这个关联可以有也可以没有，因为默认情况下，无论哪一个 updatePanel 开始回调，UpdateProgress 都会自动显示它的 ProgressTemplate 内容。若页面很复杂且有多个 UpdateProgress，则可以选择让 UpdateProgress 只关注其中的某个 UpdatePanel，这时就很有必要设置其 AssociatedUpdatePanelID 属性了。

这时运行该程序，单击"启动"按钮，还看不到进度提示信息，这是因为在后台还没有写异步回发的代码，所以该请求什么也不做，很快就返回，进度条就没有机会显示了。在按钮的单击事件中增加一个延时就可以看到进度指示了，代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
      System.Threading.Thread.Sleep(5000); //延时 5 秒
      lblmsg.Text = "处理完成";
}
```

12.3.4 使用 Timer 控件实现定时刷新

有时客户端可能想按一定时间间隔定期访问 Web 服务器以获取某些频繁更新的信息，如股票行情信息等。使用 Timer 控件可帮助你实现这种设计。

Timer 控件的使用非常简单，只要把它加入到页面上，并设置其 Interval 属性为定时时间间隔(以毫秒为单位)，那么它就会按指定的间隔不断的触发异步回发。

例如：

```
<asp:Timer ID = "Timer1" runat = "server" Interval = "1000" OnTick = "Timer1_Tick">
```

该行代码将 Timer1 的定时间隔设置为 1 秒钟，这样它每隔 1 秒便会触发一次 Timer1_Tick 事件过程，可以在该过程中编码来刷新页面。若要停止一个定时器，只需在服务器端将其 Enabled 属性设置为 false 即可。

例 12-6 在页面上显示服务器的当前时间，并能每隔 1 秒钟自动更新显示。

该示例的运行效果如图 12-7 所示。

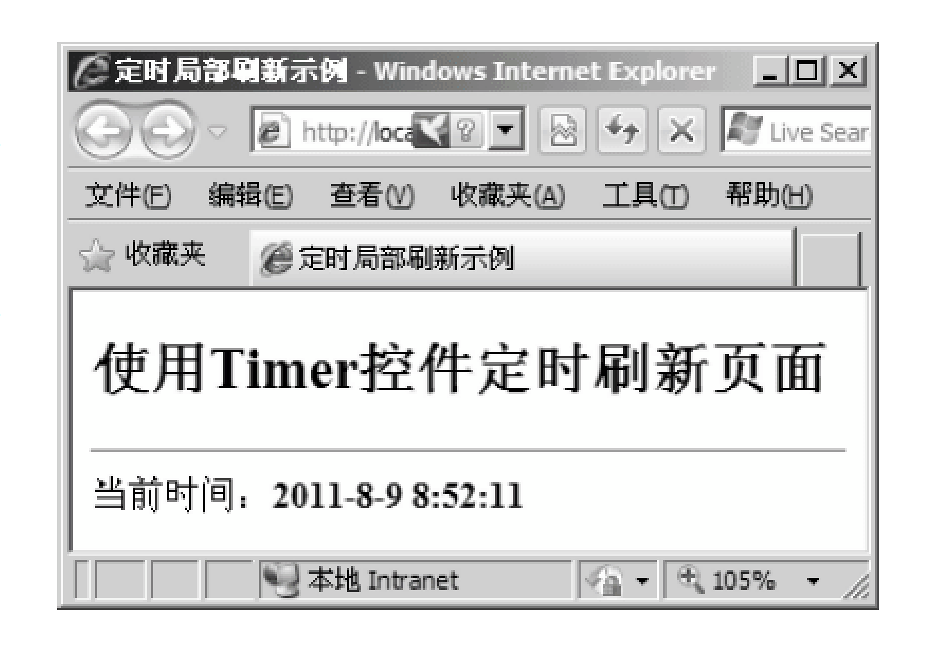


图 12-7 使用 Timer 控件更新和显示时间

该示例的页面代码如下：

```
<form id = "form1" runat = "server">
<asp:ScriptManager ID = "ScriptManager1" runat = "server"> </asp:ScriptManager>
<h2>使用 Timer 控件定时刷新页面</h2><hr />
<asp:UpdatePanel ID = "UpdatePanel1" runat = "server" UpdateMode = "Conditional">
    <ContentTemplate>
        当前时间：<asp:Label ID = "lblmsg" runat = "server" ></asp:Label>
    </ContentTemplate>
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID = "Timer1" EventName = "Tick" />
    </Triggers>
</asp:UpdatePanel>
<asp:Timer ID = "Timer1" runat = "server" Interval = "1000" OnTick = "Timer1_Tick">
</asp:Timer>
</form>
```

这里，将 Timer 控件的 Interval 属性设置为 1000 毫秒，并将其 Tick 事件注册为 UpdatePanel 的触发器。Timer1_Tick 事件过程的代码如下：

```
protected void Timer1_Tick(object sender, EventArgs e)
{
    lblmsg.Text = DateTime.Now.ToString();
}
```

可以看出，Timer 控件特别适用于部分呈现的页面，因为异步回发不会打断用户当前的工作，局部刷新也不会导致页面闪动。唯一的问题是，定时刷新会加大应用程序服务器的负载，要切实考虑清楚，将时间间隔尽可能设置的长一些。若页面上仅仅是为了显示一个时钟，就不一定非要从服务器上获取时间，最好使用 JavaScript 从客户端定时获取时间。

12.4 使用 Ajax Control Toolkit

除了 Microsoft Ajax Extensions 中提供的 4 个核心控件外，开源的 Ajax Control Toolkit 提供了大量功能强大的扩展控件，这些控件能够与现有的 Web 控件配合使用，为

Web 控件增加 AJAX 特性。

12.4.1 获取和安装 Ajax Control Toolkit

要获取 Ajax Control Toolkit 工具包，请访问 Microsoft AJAX Content Delivery Network，网址是 http://www.asp.net/ajaxlibrary/cdn.ashx。这里，不但提供了 Ajax Control Toolkit 的下载链接，还提供了各种控件的教程、示例以及编程参考等资源。

下载 Ajax Control Toolkit 的压缩包后，打开它，会看到一个名为 AjaxControlToolkit.dll 的核心程序集，以及支持各种本地语言的资源文件，另外还包含一个示例网站项目，演示各种控件的使用。

打开 Ajax Control Toolkit 的示例网站，其运行界面如图 12-8 所示。在左侧的控件列表中单击某链接，右侧窗格种就能看到该控件的运行效果及说明。

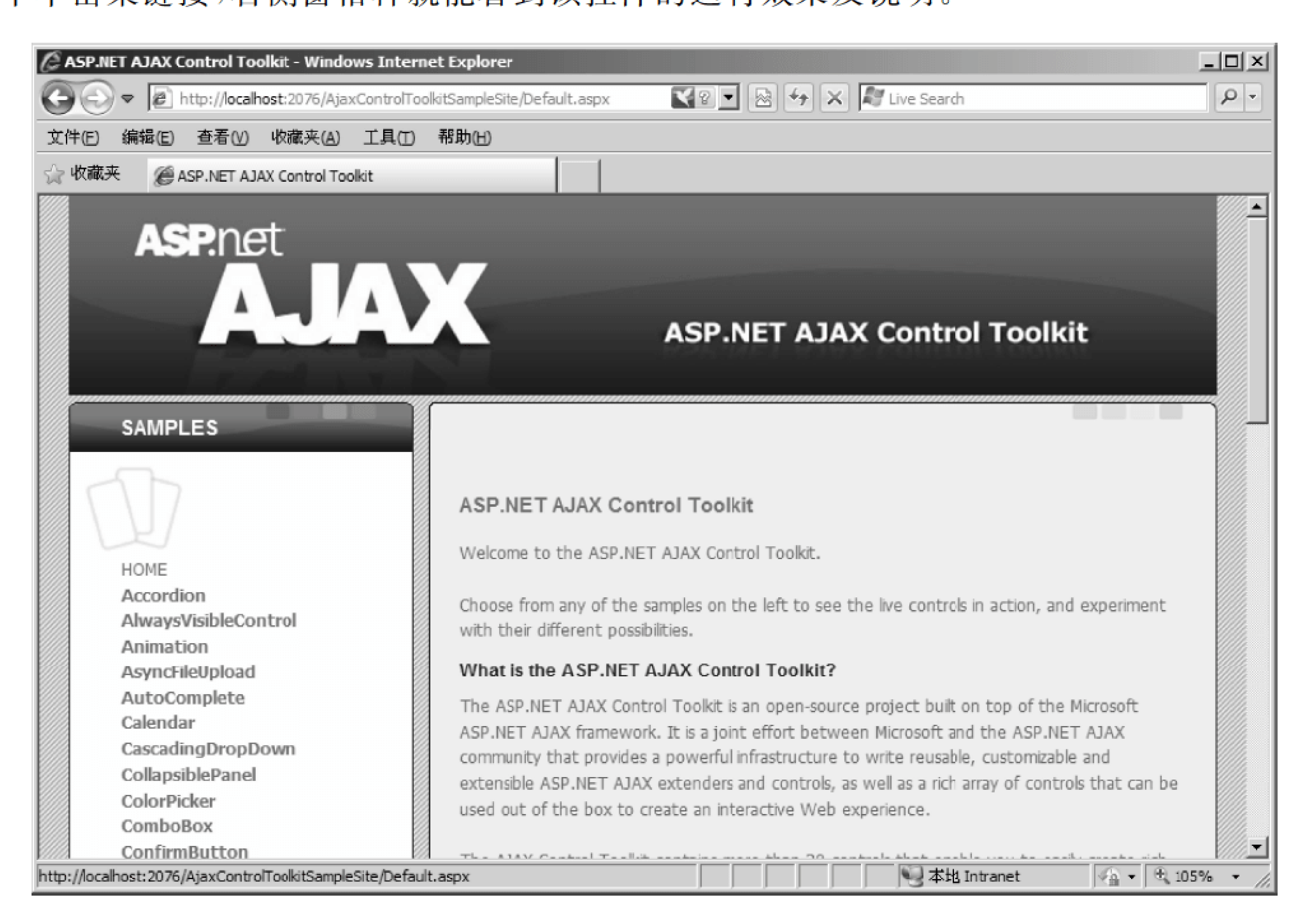


图 12-8　Ajax Control Toolkit 示例网站运行效果

为在 Web 项目中使用 Ajax Control Toolkit 中的控件，需要将相关的程序集引用到 Web 项目中，最简单的方法是将 AjaxControlToolkit.dll 程序集复制到 Web 项目的 Bin 文件夹下，并将相关语言版本的资源文件(如 zh-CHS 文件夹下为中文版资源文件)连同文件夹一起也复制到 Bin 文件夹下。

为方便开发，最好能将这些控件加入到 VS 的工具箱中，方法如下：

(1) 在工具箱上新建一个选项卡，并输入名字(如 AjaxControls)。

(2) 右击新建选项卡上的空白区域，从弹出的快捷菜单中选择“选择项”命令。

(3) 在弹出的“选择工具箱项目”对话框中，单击“浏览”按钮打开文件选择对话框，选择 AjaxControlToolkit.dll 程序集，单击“确定”按钮将这些控件添加到工具箱上。工具箱中的

Ajax 控件列表如图 12-9 所示。

12.4.2 Ajax 控件使用示例

在很多网页中，当用户输入内容时，边输入系统能够边给出一些建议，辅助用户输入，如图 12-10 所示。以往要实现这样的功能会非常复杂，有了 Ajax 控件，这项工作变得简单了。

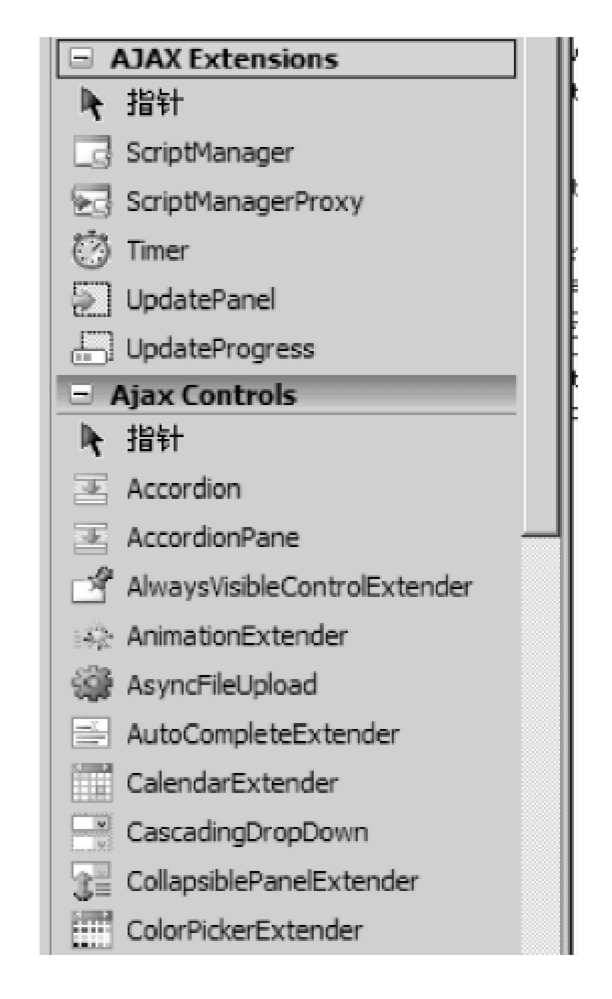




图 12-9 工具箱中的 Ajax 控件列表



图 12-10 AutoComplete 运行效果

例 12-7 使用 AutoCompleteExtender 控件实现输入自动建议功能。

当输入 Contact Name 时，能够根据输入自动显示一组建议，用户单击某建议项时，对应的文本就复制到输入框中。

(1) 建立输入页面：

从工具箱中选择一个文本框、一个 ScriptManager 控件及一个 AutoCompleteExtender 控件拖放到页面上，并配置其属性。最后生成的页面代码如下：

```
<form id = "form1" runat = "server">
    <asp:ScriptManager ID = "ScriptManager1" runat = "server"></asp:ScriptManager>
    Contact Name: <asp:TextBox ID = "txtName" runat = "server"></asp:TextBox>
    <ajaxToolkit:AutoCompleteExtender ID = "autoComplete1" runat = "server"
         TargetControlID = "txtName" ServiceMethod = "GetNames" MinimumPrefixLength = "2">
    </ajaxToolkit:AutoCompleteExtender>
    <br /><br />Enter at least two letters (such as "Fr").
</form>
```

可以看到，AutoCompleteExtender 控件的主要属性设置如下：

- TargetControlID：要辅助的目标控件。Ajax 控件基本上都是扩展控件，自身不能独立工作，必须要配合别的 Web 控件。这里是辅助名字输入框 txtName 控件。
- ServiceMethod：要调用的服务器端方法。当用户输入部分字符后，要根据输入内容向服务器端发送异步请求，查询并返回建议列表。服务器端一般通过一个 Web Service 处理该请求，这里就指定该 Web 方法的名字。
- MinimumPrefixLength：最少输入字符长度，该属性设置为 2，表示至少要输入两个

字符后才向服务器端发送异步请求。

(2) 建立 Web 方法处理异步请求：

一般情况下，可以建立专门的 Web Service 类并实现 Web 方法。但在 ASP.NET 中，为简化开发，也允许将 Web 方法定义在页面的后台代码中。本例就直接在页面中定义 Web 方法，原型代码如下：

```
[System.Web.Services.WebMethod]
[System.Web.Script.Services.ScriptMethod]
public static List<string> GetNames(string prefixText, int count)
{   …   }
```

方法声明的前面有两行标注，指明这是一个 Web 方法，并且能够被脚本调用。方法体中要根据用户输入的前缀(prefixText)，找到指定数量(count)的匹配项，并将所有匹配项以列表的形式返回。这里为提高模式匹配的效率，应该采用缓存对程序进行优化。

首先设计一个方法，能够从 Customers 表中读取所有顾客的姓名，代码如下：

```
private static List<string> GetNameListFromDB()
{
    string connectionString =
        WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
    SqlConnection con = new SqlConnection(connectionString);
    SqlCommand cmd = con.CreateCommand();
    cmd.CommandText = "SELECT ContactName FROM Customers ORDER BY ContactName";
    List<string> names = new List<string>();
    try
    {
        con.Open();
        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            names.Add((string)reader["ContactName"]);
        }
        reader.Close();
        return names;
    }
    finally
    {
        con.Close();
    }
}
```

在 GetNames 方法中，首先从缓存里面检索有没有联系人列表，若没有，则调用 GetNameListFromDB 方法获取列表并保存到缓存中。相关代码如下：

```
List<string> names = null;
// 首先检查缓存中是否有顾客姓名列表
if (HttpContext.Current.Cache["NameList"] == null)
```

```
{
        names = GetNameListFromDB();

        // 将查询数据库得到的顾客姓名列表缓存起来(保存 60 分钟)
        HttpContext.Current.Cache.Insert("NameList", names, null,
            DateTime.Now.AddMinutes(60), TimeSpan.Zero);
}
else
{
        // 若缓存已建立,则从缓存中直接得到顾客姓名列表
        names = (List<string>)HttpContext.Current.Cache["NameList"];
}
```

然后，根据输入的前缀，检索有没有符合条件的姓名，代码如下：

```
int index =-1;
for (int i = 0; i < names.Count; i++)
{
    // 若找到以该前缀开始的姓名,则得到第一个匹配项的索引号
    if (names[i].StartsWith(prefixText, StringComparison.InvariantCultureIgnoreCase))
    {
            index = i; break;
    }
}
// 若没有找到匹配项,则直接向客户端返回空的列表
if (index ==-1) return new List<string>();
```

当找到第一匹配的姓名后，还要继续找到其他匹配项一起返回。由于列表中所有项已经按顾客姓名排好了序，找到第一个后，其他匹配记录就紧随其后，所以比较起来很方便。在缓存中匹配并返回数据的代码如下：

```
List<string> wordList = new List<string>();
for (int i = index; i < (index + count); i++)
{
    // 若已到达列表结尾,则跳出
    if (i >= names.Count) break;
    // 若该项的姓名已经不匹配,则跳出
    if (!names[i].StartsWith(prefixText, StringComparison.InvariantCultureIgnoreCase))
        break;
    // 将匹配的姓名加入到返回列表中
    wordList.Add(names[i]);
}
return wordList;
```

关于其他 Ajax 控件的使用方法，请参考工具包中的示例网站。

Microsoft Ajax 技术提供了一个多层的平台，可以在多个层次上使用 ASP.NET AJAX 框架：

(1) 仅使用客户端模块的功能，编写自己的 JavaScript 或调用库中的 JavaScript，而服

务器端暴露 Web 方法供 JavaScript 调用。

(2) 使用 Ajax Extensions 提供的基本控件,将传统 Web 控件组织到 UpdatePanel 上来实现异步请求与局部刷新。

(3) 使用扩充的 Ajax Control Toolkit 控件,配合传统 Web 控件,开发功能更强大的 AJAX 应用。

(4) 创建自己的客户端组件,结合自定义的服务器端组件开发 AJAX 应用。

12.5 习题和上机练习

简答题

(1) 什么是 AJAX? 为什么要使用 AJAX? 谈一下你对 AJAX 的认识。

(2) AJAX 应用和传统 Web 应用相比,有哪些主要的不同?

(3) 在 AJAX 体系结构中,XMLHTTPREQUEST 对象的作用是什么? 如何创建该对象?

(4) 试简要介绍 XMLHTTPREQUEST 对象的常用方法和属性。

(5) 试总结 AJAX 的技术体系中主要包含了哪些核心技术。

(6) 试总结 AJAX 技术的优点和缺点。

参 考 文 献

[1] 金旭亮. ASP.NET 程序设计教程 . 北京：高等教育出版社，2009.
[2] 金旭亮. .NET 2.0 面向对象编程揭秘 . 北京：电子工业出版社，2007.
[3] FREEMAN E. Head First HTML with CSS & XHTML . 影印版. 南京：东南大学出版社，2006.
[4] RICHTER J. Microsoft .NET 框架程序设计. 李建中译. 武汉：华中科技大学出版社，2004.
[5] ESPOSITO D. ASP.NET 2.0 高级编程. 施平安译. 北京：清华大学出版社，2006.
[6] ESPOSITO D. ASP.NET 2.0 技术内幕. 施平安译. 北京：清华大学出版社，2006.
[7] 开发人员中心. http://msdn. microsoft. com/zh-cn/netframework/. .NET Framewotk.
[8] W3Schools Online Web Tutorials. http://vwww. w3schools. com/.
[9] 官方网站. http://www. asp. net/. Microsoft asp. net.
[10] 金雪云. ASP.NET 简明教程(C#篇). 北京：清华大学出版社，2007.